UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Processamento Automático de Requisitos de Software Crítico, Expressos em Linguagem Natural

Leopoldo José Relvas Ismael

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor João Paulo Cordeiro
Co-Orientador: Prof. Doutor Simão Melo de Sousa

Covilhã, outubro de 2014

Dedicatória

Quero dedicar esta dissertação aos meus pais e família
Quer pela motivação quer pelos valores transmitidos ao longo da vida
Sem a intervenção deles a realização desta dissertação não teria sido possível.

Agradecimentos

Em primeiro lugar, e como não poderia evitar, gostaria de agradecer aos meus pais pelo apoio, paciência, incentivo e confiança que me transmitiram ao longo de todo o meu percurso académico bem como ao longo de toda a vida. Sem a dedicação, incentivo e o esforço deles não teria sido possível atingir este objetivo.

Quero também agradecer à minha amiga e namorada, Raquel Lebre, pela motivação, paciência e carinho dado durante não só durante a realização da dissertação, mas também ao longo de todo o percurso académico. Mesmo estando longe, esteve sempre presente para dar força e motivação para continuar a lutar pelos meus objetivos.

Ao Prof. Doutor Simão Melo de Sousa estou profundamente agradecido pela oportunidade que me foi dada ao integrar o projeto ***Platform for Software Verification and Validation*** (PROVA) que em conjunto com o Prof. Doutor João Paulo Cordeiro proporcionaram a junção de duas áreas em que tive muito gosto em trabalhar e que cuja orientação por parte dos dois foi indispensáveis ao longo das várias fases deste projeto. Os seus conhecimentos, observações, prontidão e tempo despendido foram de grande importância para a realização desta dissertação ao longo de todas as etapas.

Ao Joaquim Tojal e José Faria da ***EDUCED*** pelo interesse e apoio dado ao longo do desenvolvimento do projeto e dissertação.

Quero expressar os meus agradecimentos ao meu colega do grupo de investigação, Tiago Carvalho, tanto pela ajuda na interpretação de conceitos, *brainstorming* e colaboração no âmbito do projeto PROVA.

Gostaria de agradecer também aos meus grandes amigos, João Gouveia, João Silveira, Pedro Querido, Manuel Meruje, Samuel Dias e Ruben Santo, pelo ânimo, distração e companhia nas noites de trabalho.

Por último gostaria de agradecer aos restantes colegas do ***RELiablE And SEcure Computation Group*** (RELEASE) pelas interações que muitas vezes fizeram surgir ideias que facilitaram o desenvolvimento desta dissertação.

"Se soubessemos o que estamos a fazer não se chamaria investigação." - Albert Einstein

Resumo

A presente dissertação resulta da necessidade de classificação de um conjunto de requisitos de software crítico não categorizados, em categorias distintas através de métodos tradicionais e de ferramentas de *Natural Language Processing*, no contexto do projeto PROVA - *Platform for Software Verification and Validation*, levado a cabo pela empresa EDUCED.

Justifica-se este projeto com a necessidade de existir um pré-processamento do texto referente aos requisitos de software crítico que a plataforma do PROVA irá suportar, mas também devido ao facto de a classificação dos requisitos de software ser um trabalho manual, feito por um engenheiro de requisitos podendo levar bastante tempo a ser concluído.

Nesta dissertação é apresentado, detalhadamente, um método para a criação de um classificador de requisitos de software crítico baseado em métodos de processamento de texto, complementados com ferramentas de processamento de linguagem natural para uma classificação mais precisa. O sistema irá permitir a entrada dos requisitos de software crítico em vários formatos e retornará no final a classificação individual de cada requisito.

A classificação individual do requisito é obtida através do recurso a um *parser* que recebe o texto do requisito de software e procura palavras-chave e/ou expressões que auxiliem na identificação de cadeias de símbolos, que de acordo com a gramática definida e através de métodos de comparação dessas cadeias irá concluir em qual categoria o requisito pode ser classificado. De referir ainda que quando o texto do requisito de software chega ao *parser*, foi anteriormente complementado com o etiquetas sintáticas criadas para uma melhor classificação do requisito de software.

Palavras-chave

Requisitos de Software, Classificador, Processamento de Linguagem Natural, Parser, Gramática, Etiquetas Sintáticas

Abstract

The present dissertation results from the need of classifying a set of non categorized critical software requirements in different categories using traditional methods and Natural Language Processing tools, under the PROVA project - *Platform for Software Verification and Validation*, carried out by EDUCED company.

This project is justified by the need of a preprocessing of the text referring to the critical software requirements PROVA platform will support, but also due to the fact that the classification of software requirements is a manual task performed by a requirements engineer that can take a long time to complete.

In this dissertation will be presented in detail a method to create a classifier of critical software requirements with software-based methods of text processing, supplemented with natural language processing tools for a more accurate classification. The system will allow the entry of critical software requirements in various formats and will return at the end, the individual classification of each software requirement.

The individual classification of the software requirement is obtained by using a parser that receives the software requirement text and searches for keywords and/or sentences that assist in the identification of strings of symbols, which according to the defined grammar and using string comparison methods will conclude in which category the software requirement fit. Also note that when the text reaches the software requirement parser, it was previously complemented with syntactic labels designed for better software requirement classification.

Keywords

Software Requirements, Classifier, NLP, Parser, Grammar, Syntactic Tags

Índice

Dedicatória iii

Agradecimentos v

Resumo vii

Abstract ix

Índice xi

Lista de Figuras xii

Lista de Tabelas xv

Lista de Acrónimos xvii

1 Introdução 1
- 1.1 Enquadramento 1
- 1.2 Motivação 2
- 1.3 Objetivos 2
- 1.4 Abordagem 3
- 1.5 Contribuições 3
- 1.6 Estrutura da Dissertação 3

2 Estado da Arte 5
- 2.1 Introdução 5
- 2.2 Engenharia de Requisitos 5
  - 2.2.1 Elicitação dos Requisitos 6
  - 2.2.2 Modelação e Análise dos Requisitos 7
  - 2.2.3 Comunicação dos Requisitos 9
  - 2.2.4 Consensualização dos Requisitos 9
  - 2.2.5 Evolução dos Requisitos 9
- 2.3 Classificação de Requisitos 10
  - 2.3.1 Requisitos Funcionais 10
  - 2.3.2 Requisitos Não Funcionais 11
- 2.4 Principais Desafios 12
- 2.5 Soluções Existentes 13
- 2.6 Métodos Desenvolvidos 14
- 2.7 Conclusões 14

3 Implementação dos Algoritmos 15
- 3.1 Introdução 15
- 3.2 POS Tagger 15
  - 3.2.1 Implementação 17
- 3.3 Classificação de Requisitos 18
  - 3.3.1 Classificador Baseado em Métodos Formais e PLN 18

3.3.2 Classificador Baseado em Métodos Estatísticos 23
3.4 Conclusões 26

4 Arquitetura 27
4.1 Introdução 27
4.2 Cenários de Utilização 27
4.3 Descrição da Biblioteca 27
4.3.1 Classe *Result.cs* 28
4.3.2 Classe *RPIException.cs* 28
4.3.3 Classe *GramaticaFuncional.cs* 28
4.3.4 Classe *POSTagger.cs* 28
4.3.5 Classe *Main.cs* 29
4.3.6 Classe *Filter.cs* 29
4.4 Ficheiros Auxiliares e Dependências 30
4.4.1 Ficheiros Auxiliares 30
4.4.2 Dependências 31
4.5 Conclusões 31

5 Resultados 33
5.1 Introdução 33
5.2 Avaliação de Desempenho 33
5.2.1 Classificador Baseado em Métodos Formais e PLN 33
5.2.2 Classificador Baseado em Métodos Estatísticos 33
5.3 Conclusões 35

6 Conclusão e Trabalho Futuro 37
6.1 Conclusão 37
6.1.1 Conclusões 37
6.2 Trabalho Futuro 38
6.2.1 Aspetos de Implementação 38
6.2.2 Novas Implementações 39

Referências 41

A Anexos 45
A.1 Anexo A 45
A.2 Anexo B 47
A.3 Anexo C 48
A.4 Anexo D 48

Lista de Figuras

2.1 Esquema das subdivisões dos Requisitos Não Funcionais 11

3.1 Esquemática do funcionamento do *Stanford POS Tagger* 16
3.2 Janela principal do *Irony Grammar Explorer* 22
3.3 Funcionamento do Classificador Baseado em Métodos Formais e PLN 23
3.4 Exemplo de classificação usando o Classificador Baseado em Métodos Formais e PLN 23
3.5 Exemplo de classificação usando o Classificador Baseado em Métodos Estatísticos 26

5.1 Resultado final do Classificador Baseado em Métodos Formais e PLN 34
5.2 Resultado final do Classificador Baseado em Métodos Estatísticos 34

Lista de Tabelas

2.1 Tabela comparativa das ferramentas usadas na ER. 14

3.1 Tabela com a listagem das etiquetas POS usadas no Penn Treebank 17

4.1 Exemplo das primeiras linhas das listas com as palavras mais relevantes 31
4.2 Tabela com a listagem das bibliotecas necessárias para o correto funcionamento dos métodos de classificação 31

Lista de Acrónimos

UBI	Universidade da Beira Interior
PLN	Processamento de Linguagem Natural
PROVA	Platform for Software Verification and Validation
ER	Engenharia de Requisitos
RNF	Requisitos Não-funcionais
RR	Rastreabilidade dos Requisitos
TRS	Technical Requirements Specification
POS	Part of Speech
JVM	Java Virtual Machine
GPLEX	Gardens Point LEX
GPPG	Gardens Point Parser Generator

Capítulo 1

Introdução

Com a evolução das capacidades da tecnologia, o constante crescimento das necessidades tecnológicas e com a exigência operacional dos ser humanos em relação aos sistemas tecnológicos, as empresas foram forçadas a rever o processo de desenvolvimento e manutenção do seu software visto que, segundo um estudo [GGP01] 79% do orçamento das empresas de desenvolvimento de software são gastos em manutenção dos mesmos e que cerca de 36% do pessoal interno está focado no suporte e manutenção do sistema.

Desta forma, um bom planeamento da arquitetura e das funcionalidades do sistema pode levar a uma grande redução de custos com a manutenção de software, assim sendo a definição de requisitos de software torna-se um passo importante no processo de desenvolvimento de qualquer sistema. A existência de requisitos de software, que no fundo atuam como diretrizes, assistindo e guiando os programadores na maneira de desenvolver um certo componente do sistema, ou como o sistema A transporta informação ao sistema B, ou uma definição que exige que o sistema responda de uma forma específica a determinado problema, exige que todo o processo respeite um arquitetura previamente estruturada e analisada de modo a que no futuro não seja necessário uma manutenção tão dispendiosa.

Com a dimensão dos sistemas de software a crescer cada vez mais, o número de requisitos sobe exponencialmente, o que pode levar a um tratamento dos mesmos mais demorado, aumentando inevitavelmente o tempo do seu processo de desenvolvimento. Existe cada vez mais a necessidade de ser eficientemente produtivo, e uma melhor organização dos requisitos, através duma classificação automatizada dos mesmos, pode levar à redução do tempo usado pelo engenheiro de requisitos na organização dos requisitos.
As ferramentas criadas no âmbito desta dissertação têm como objetivo o aumento da eficiência das ferramentas atuais de gestão e armazenamento de requisitos, ao mesmo tempo que ajudam os engenheiros de requisitos a desempenhar o seu papel no processo de desenvolvimento de um software.

1.1 Enquadramento

Devido à grande quantidade de requisitos de software que o engenheiro de requisitos tem de analisar, tornou-se necessário uma organização dos mesmos, através da atribuição de categorias aos requisitos. A classificação mais básica define duas categorias de requisitos de software (serão explicados em detalhe no Capítulo 3), são elas:

1. funcionais, o que o sistema deve fazer;
2. não-funcionais, considerados como atributos dos requisitos funcionais.

Este tipo de classificação é feita em primeiro lugar, para separar o ambiente das funcionalidades, o que o sistema têm de fazer para estar de acordo com as necessidades dos interessados,

do ambiente de design, como o sistema desempenha essas funcionalidades e de que maneira terá de responder a certas situações, com a intenção de dar aos designers liberdade total para explorar qualquer tipo arquitetura para projeto. Ao limitar os requisitos num ambiente puramente funcional, concentramo-nos no que o cliente realmente definiu, permitindo às pessoas envolvidas usarem todo o seu conhecimento e criatividade como suporte ao desenvolvimento. A separação do ambiente funcional do ambiente de design é fundamental na engenharia de software, porque torna possíveis todo o tipo de heurística para encontrar bons designs [Kov03]. Em segundo lugar, os requisitos funcionais são distintos dos requisitos não-funcionais, porque são, geralmente, referentes a problemas muito mais difíceis de projetar e testar do que os requisitos não-funcionais. Conseguir que um design corresponda aos requisitos não-funcionais, muitas vezes ocupa muito tempo e envolve problemas complicados. Os requisitos funcionais tendem a ser mais simples, por exemplo, o desenho e implementação de uma base de dados para armazenamento de informação é uma tarefa bastante simples, no entanto obter alto desempenho numa rede distribuída com alta carga de transações requer muito mais planeamento, design e testes [Kov03].

1.2 Motivação

O presente trabalho é o resultado de uma parceria entre o grupo de investigação *RELiable and SEcure Computation* (RELEASE) da Universidade da Beira Interior (UBI) e a empresa EDUCED no âmbito do projeto PROVA - Platform for Software Verification and Validation. Este projeto envolve vários trabalhos para além da presente dissertação e tem como objetivo a criação de um método para classificação de requisitos de software crítico para possível utilização pela empresa EDUCED.

Esta dissertação nasce da necessidade cada vez maior de haver uma ferramenta automatizada que permita não só a gestão mas também o tratamento automatizado dos requisitos de software referentes a determinado projeto. A maior parte da ferramentas existentes no mercado só permite o armazenamento e gestão dos requisitos mas não a sua categorização, sendo esse o foco desta dissertação. Existe também uma motivação pessoal para o desenvolvimento desta dissertação por parte do autor, uma vez que sempre foi interessado pela área de processamento de linguagem natural e pela vontade de contribuir com algo inexistente no mercado referente aos requisitos.

1.3 Objetivos

O principal objetivo desta dissertação é a criação de um método que permita a classificação de um conjunto de requisitos de software crítico não categorizados, em categorias distintas através de métodos convencionais e de ferramentas de Processamento de Linguagem Natural (PLN). É pretendido que no final exista uma biblioteca com as ferramentas usadas para posterior implementação no núcleo do PROVA - *Platform for Software Verification and Validation*.
Os objetivos principais previstos no plano de trabalho são os seguintes:

- Desenvolvimento de um estudo sobre trabalho relacionado e técnicas existentes de processamento de linguagem natural;

- Avaliação e comparação de tecnologias e ferramentas existentes, evidenciando as principais vantagens e limitações;
- Propor um conjunto de técnicas a implementar que permita a interpretação semântica do conteúdo dos requisitos;
- Desenvolver e implementar as técnicas propostas para integração no PROVA.

Espera-se que o produto final satisfaça os objetivos e as necessidades requeridas, de forma a merecer a sua implementação na ferramenta para o qual foi desenhado.

1.4 Abordagem

Este projeto foi abordado por fases, o trabalho realizado pode ser dividido em duas fases principais, a primeira fase, a fase de Investigação e a segunda a de Desenvolvimento.

A fase de Investigação pode ainda ser subdividida em investigação teórica, onde foram estudados conceitos na área da engenharia de requisitos, incidindo principalmente na análise e tratamento de requisitos, e em investigação técnica, onde a investigação incidiu nos conceitos, componentes e nas funcionalidades de ferramentas de interpretação de texto, nomeadamente *parsers*, métodos de etiquetagem sintática de texto e métodos estatísticos de processamento de linguagem natural de forma a perceber a viabilidade e as potencialidades dos métodos a desenvolver.

Na fase de Desenvolvimento foram implementados dois classificadores de requisitos de software, assim como estudadas tecnologias e métodos para a criação dos mesmos. Um dos classificadores é baseado em métodos formais e PLN, enquanto que o outro é baseado em métodos estatísticos.

1.5 Contribuições

No âmbito desta dissertação foram concluídos com sucesso duas ferramentas de classificação de requisitos de software. Estes dois métodos de classificação desenvolvidos estarão disponíveis para utilização através de uma biblioteca que terá todas as suas funcionalidades disponíveis ao utilizador final.

1.6 Estrutura da Dissertação

Este documento está dividido em vários capítulos que mostram sequencialmente o processo de investigação desenvolvido. Neste primeiro capitulo foi descrito o projeto, as motivações, os objetivos que se pretendem alcançar, bem como as contribuições do projeto.

Capítulo 2 - Estado da Arte
Neste capítulo apresenta-se uma introdução à engenharia de requisitos, especificamente à área que trata da elicitação e análise de requisitos de software, uma descrição da mesma, conceitos e métodos por ela trabalhados. Este capítulo pretende fornecer uma visão para quem não esteja

dentro da problemática tratada neste trabalho.

Capítulo 3 - Algoritmos Implementados

O terceiro capítulo apresenta os métodos implementados, os passos necessários para a implementação e uma análise dos métodos.

Capítulo 4 - Arquitetura

O quarto capítulo aborda a arquitetura tecnológica seguida neste trabalho de uma forma geral. Este capítulo apresenta a estrutura na qual assenta a implementação do sistema criado bem como a justificação das soluções apresentadas de forma a fornecer uma visão do funcionamento de todo o sistema em cada uma das fases.

Capítulo 5 - Resultados

O quinto capítulo apresenta os resultados obtidos no sistema implementado, as suas funcionalidades e características.

Capítulo 6 - Conclusão e Trabalho Futuro

O último capítulo apresenta as conclusões sobre o trabalho desenvolvido e descreve funcionalidades que podem ser futuramente integradas no sistema de forma a complementar a solução.

Capítulo 2

Estado da Arte

2.1 Introdução

Neste capítulo apresenta-se uma introdução à engenharia de requisitos (ER), disciplinas que constituem as bases para uma ER eficaz, uma breve descrição do contexto e das bases necessárias para iniciar um processo de ER e uma descrição das atividades centrais da ER, pretendendo fornecer uma visão geral para quem não esteja dentro da problemática tratada nesta dissertação.

2.2 Engenharia de Requisitos

O principal fator que mede o sucesso de um sistema de software é o grau em que ele se encontra em concordância com o propósito para o qual foi desenvolvido. Em termos gerais, a engenharia de requisitos de sistemas de software (ER) é o processo de descobrir a intenção para a qual o sistema de software vai ser construído, identificando as partes interessadas e suas necessidades, documentando estes de uma forma a ser possível uma análise, para uma posterior implementação mais canalizada.

Podem identificar-se as seguintes atividades da ER como essenciais:

- elicitação dos requisitos;
- modelação e análise dos requisitos;
- comunicação dos requisitos;
- consensualização dos requisitos;
- evolução dos requisitos.

É de todo vantajoso analisar o papel que a ER tem no software e na engenharia de sistemas, e as disciplinas que a compõem. Segundo [Zav97] ER define-se como o ramo da engenharia de software preocupada com os objetivos reais das funções e limitações dos sistemas de software. Ela preocupa-se também com as relações destes fatores para chegar a especificações precisas de comportamentos do software, e sua evolução ao longo do tempo e reutilização.
Esta definição é boa por várias razões. Em primeiro lugar, destaca a importância de objetivos reais que motivam o desenvolvimento de um sistema de software. Estes representam o porquê? e o o quê? de um sistema de software. Em segundo lugar, refere-se a especificações precisas, que fornecem a base para a análise dos requisitos, validando que eles são, na verdade, o que o cliente quer, definem o que os designers têm de construir, e permitem verificar que o sistema foi construído corretamente no momento da entrega. Por fim, a definição refere-se a especificações evolução ao longo do tempo e reutilização, enfatizando a realidade de um mundo em mudança e a necessidade de reutilizar especificações parciais, como acontece em muitos outros

ramos da engenharia. Seja vista ao nível do sistema ou a nível de software, ER é um processo multidisciplinar centrado no ser humano, onde as ferramentas e técnicas utilizadas recorrem a uma grande variedade de áreas e espera-se que o engenheiro de requisitos domine um conjunto de habilidades de diferentes disciplinas.

Serão abordadas de seguida as atividades principais que servem de ferramenta à ER, serão descritas individualmente realçando a sua função e importância no processo de ER.

2.2.1 Elicitação dos Requisitos

A elicitação de requisitos é a atividade considerada, normalmente, como o primeiro passo no processo de ER. O termo elicitação é preferido em relação a captura, para evitar a interpretação de que dos requisitos estão algures à espera para recolha e que para tal basta simplesmente fazer as perguntas certas [Gog94]. Informações recolhidas durante a elicitação de requisitos, muitas vezes têm que ser interpretadas, analisadas, modeladas e validadas antes do engenheiro de requisitos dizer que foi encontrado um conjunto suficiente de requisitos de sistema para iniciar o desenvolvimento. A elicitação de requisitos está muito relacionada com outras atividades ER - em grande parte, a técnica de elicitação utilizada é impulsionada pela escolha do esquema de modelação, e vice-versa: muitos esquemas de modelação implicam a utilização de determinadas técnicas de elicitação.

Um dos objetivos mais importantes da elicitação é descobrir qual o problema que precisa de ser resolvido, ou seja, identificar os limites do sistema. Esses limites definem, num alto nível, onde o sistema final se vai encaixar no ambiente operacional. A identificação e definição de limites de um sistema afeta todos os esforços de elicitação subsequentes. A identificação das classes de utilizadores, de objetivos e tarefas, de cenários e de casos de uso depende de como os limites foram definidos.

A identificação dos interessados, definidos como os indivíduos ou organizações que beneficiam ou perdem com o sucesso ou fracasso de um sistema é também importante. Os interessados incluem clientes, que pagam pelo sistema, as equipas de desenvolvimento, que projetam, constroem e mantêm o sistema, e os utilizadores finais, que interagem com o sistema para desempenhar as suas funções. No que diz respeito a sistemas interativos, os utilizadores têm um papel fulcral no processo de elicitação, uma vez que a usabilidade do sistema só pode ser definida nos termos da população-alvo a que se destina. Os próprios utilizadores não são homogéneos, e faz parte do processo de elicitação identificar as necessidades das diferentes classes de utilizadores, como os novos utilizadores, utilizadores experientes, utilizadores ocasionais, utilizadores com deficiência, entre outros [SFG99].

Metas definem os objetivos que um sistema deve atingir. A identificação de metas a alto nível no início do processo de desenvolvimento é crucial, no entanto a elicitação de requisitos orientada a metas [DvLF93] é uma atividade contínua, isto é, à medida que o desenvolvimento prossegue, as metas de alto nível (por exemplo, metas de negócio) são refinadas em metas de nível mais baixo (por exemplo, metas mais técnicas que, eventualmente, são implementadas num sistema). A identificação de metas concentra o engenheiro de requisitos no domínio do problema e as necessidades dos interessados e não em possíveis soluções para esses problemas. Existem várias técnicas de identificação de requisitos e a escolha da técnica a usar depende

bastante do tempo e dos recursos disponíveis ao engenheiro de requisitos, e naturalmente, do tipo de informação que precisa ser recolhida. Distinguem-se as principais técnicas de elicitação:

- Técnicas tradicionais incluem uma ampla classe de técnicas genéricas de recolha de dados, como a utilização de questionários e pesquisas, entrevistas e análise de documentação existente tais como organogramas, de processos modelo ou padrão e manuais de utilizador ou outros de sistemas já existentes;
- Técnicas de elicitação em grupo visam incentivar acordo entre os interessados, enquanto tira partido da dinâmica de equipa para compreender melhor as necessidades. Estas técnicas incluem o *brainstorming* e o uso de grupos de foco [MR96];
- Prototipagem é usada para elicitação onde existe um grande grau de incerteza sobre os requisitos, ou onde a opinião dos interessados é necessária mais cedo [Dav92]. A técnica da prototipagem pode também ser facilmente combinada com outras técnicas, por exemplo, através do uso de um protótipo para provocar discussão numa técnica de elicitação em grupo, ou ainda como base para um questionário;
- As técnicas orientadas em modelos fornecem um modelo específico do tipo de informações a serem recolhidas, e usam este modelo para conduzir o processo de elicitação;
- Técnicas cognitivas incluem uma série de técnicas originalmente desenvolvidas para a aquisição de conhecimento para sistemas baseados em conhecimento [SG96]. Essas técnicas incluem a análise de protocolos (em que um especialista pensa em voz alta durante a execução de uma tarefa, de modo a proporcionar ao observador ideias sobre os processos cognitivos utilizados na realização da tarefa), *laddering* que usa teste para avaliar a estrutura e nível do conhecimento dos interessados, *card sorting* onde é pedido aos interessados para classificar grupos de cartões, em que cada um dos quais tem nome de alguma entidade e normalmente são ordenadas de acordo com a importância para os interessados, reportório em grelha que implica a construção de uma matriz de entidades, pedindo aos interessados para atribuir atributos aplicáveis às entidades e valores para as células em cada entidade;
- As técnicas contextuais surgiram na década de 1990 como uma alternativa para as técnicas tradicionais e técnicas cognitivas [GL93]. Estas incluem a utilização de técnicas etnográficas como a observação dos participantes. Elas incluem também a etnometedologia e análise de conversações, os quais se aplica a análise granular para identificação de padrões na conversa e interação [VS99].

Com a grande variedades de técnicas de elicitação de requisitos disponíveis para o engenheiro de requisitos utilizar, é necessária orientação sobre o seu uso. Os métodos fornecem uma forma de prestar essa orientação. Cada método em si tem seus pontos fortes e fracos, e normalmente, adequa-se ao uso conforme os domínios. É claro que, em algumas circunstâncias, um método rígido pode nem ser necessário. Em vez disso, o engenheiro de requisitos precisa simplesmente de selecionar a técnica ou técnicas mais apropriadas [MR96].

2.2.2 Modelação e Análise dos Requisitos

Define-se modelação como a construção de descrições abstratas que são passíveis de interpretação sendo uma atividade fundamental na ER. Os modelos podem ser utilizados para representar toda uma gama de produtos do processo de ER. Além disso, em muitas abordagens da modelação

são utilizados como ferramentas de elicitação, onde os modelos parciais produzidos são usados como ferramenta para impulsionar a recolha de informações. De seguida vão ser descritos alguns métodos de modelagem gerais usados na ER.

A maioria das atividades de ER tem como contexto uma organização em que o desenvolvimento de um novo sistema de software ocorre sobre um sistema existente. A modelação e análise empresarial implica a compreensão da estrutura de uma organização, das regras de negócios que afetam o seu funcionamento, das metas, tarefas e responsabilidades dos seus membros e dos dados que ela gera e manipula. A modelação empresarial é muitas vezes usada para encontrar o objetivo principal de um sistema, descrevendo o comportamento da organização em que esse sistema irá operar [LK95]. A modelação de metas é particularmente útil na ER, metas de negócio de alto nível podem ser refinadas várias vezes, como parte do processo de elicitação, levando à definição de requisitos que podem ser depois operacionalizados [DvLF93].

Os grandes sistemas baseados em computadores, especialmente os sistemas de informação usam e geram grandes volumes de informação que, precisa ser entendida, manipulada e controlada. Decisões têm de ser feitas sobre as informações que o sistema terá que representar, e como a informação que o sistema possui corresponde à real. A modelação de dados oferece a oportunidade para abordar estas questões na ER. Tradicionalmente, a modelação Entidade-Relacionamento-Atributo (ERA) é usada para este tipo de modelação e análise. No entanto, a modelação orientada a objetos, que utiliza classes e objetos hierárquicos, estão cada vez mais a substituir a modelação ERA.

A modelação de requisitos envolve muitas vezes modelação do comportamento dos interessados e dos sistemas, tanto dos existentes como dos a desenvolver. A distinção entre a modelação de um sistema existente, e modelação de um sistema a implementar é importante, e é muitas vezes dificultada pelo uso das mesmas técnicas de modelagem para ambos. Métodos de análise inicial sugerem que se deve começar por modelar a forma como o trabalho é realizado atualmente, a análise desta situação serve para determinar a funcionalidade essencial, e finalmente, construir o modelo de como o novo sistema deverá operar. Uma ampla gama de métodos de modelação está disponível, como métodos orientados a objetos e também métodos formais. Esses métodos fornecem diferentes níveis de precisão e são passíveis de diferentes tipos de análise. Os métodos formais, por exemplo, podem ser difíceis de construir, mas são bastante acessíveis a uma análise automatizada [Saa97].

Uma grande parte do processo de ER é a criação de descrições de domínio [JZ93]. Um modelo de domínio fornece uma descrição abstrata em que o sistema planeado irá operar. A construção de modelos de domínio explícitos oferece duas vantagens principais: permitem raciocínio detalhado do o que é assumido sobre o domínio, e fornecem oportunidades para a reutilização de requisitos dentro do mesmo domínio.

Requisitos não-funcionais (RNF), também conhecidos como requisitos de qualidade, são geralmente mais difíceis de expressar de uma forma quantificável, tornando-os mais difíceis de analisar. Em particular, RNF tendem a ser as propriedades de um sistema como um todo, e, consequentemente, não podem ser verificados nos componentes individuais. Há um crescente aumento de grupos de pesquisa focados em determinados tipos de RNF, como *safety* [LHM+98, MLR+97], a *security* [Chu93], a *reliability* [GNCC98], e *usability* [Joh92].

Uma das principais vantagens da modelação de requisitos é a oportunidade que esta oferece para os analisar. Técnicas de análise têm sido investigadas na ER incluindo animação de requisitos [GH96], raciocínio automatizado conhecimento [FN88] e a verificação de consistência [Hol97].

2.2.3 Comunicação dos Requisitos

ER não é apenas um processo de descoberta e especificação de requisitos, é também um processo para facilitar a comunicação eficaz desses mesmos requisitos entre os diferentes interessados. A forma como os requisitos são documentados desempenha um papel importante na garantia de que eles podem ser lidos, analisados, reescritos e validados.

O que é cada vez mais definido como crucial no processo de ER, é gestão de requisitos. Define-se gestão de requisitos como a capacidade, não só para escrever os requisitos, mas fazê-lo também de uma forma que sejam legíveis e detetáveis, a fim de gerir a sua evolução ao longo do tempo.

A rastreabilidade dos requisitos (RR) é outro fator importante que determina o quão fácil é ler, alcançar, consultar e alterar a documentação de requisitos. Define-se RR como a capacidade de descrever e seguir a vida de um requisito em ambas as direções, ou seja, desde as suas origens, através do seu desenvolvimento e especificação, para implantação e posterior utilização, e através de todos os períodos de melhoramento e iteração em qualquer uma destas fases [GF94]. RR está no centro da prática da gestão de requisitos na medida em que pode fornecer uma justificação para os requisitos e é a base para as ferramentas que analisam as consequências e o impacto das mudanças.

2.2.4 Consensualização dos Requisitos

Como os requisitos são elicitados e modelados, manter acordo entre todos os interessados pode ser um problema, especialmente quando os interessados têm objetivos divergentes. Recordando que a validação é o processo que estabelece que os requisitos e os modelos elicitados têm de fornecer um relato exato das necessidades dos interessados, descrever explicitamente os requisitos é uma condição necessária para a resolução de conflitos entre as partes interessadas.

A validação de requisitos é difícil por dois motivos. O primeiro motivo é de natureza filosófica, e diz respeito à questão da verdade e o que se pode conhecer. A segunda razão é social, e diz respeito à dificuldade de se chegar a um acordo entre os diferentes intervenientes com conflitos entre objetivos. Estes motivos são agravados por uma série de questões contextuais, incluindo questões contratuais, e o facto de que o ambiente político e social em que a introdução de um novo sistema informático muda a natureza do trabalho e das organizações [Leh80].

2.2.5 Evolução dos Requisitos

Os sistemas informáticos bem sucedidos estão sempre a evoluir à medida que o ambiente em que operam altera ou os requisitos dos interessados mudam. Mudanças na documentação de requisitos precisam ser geridas. No mínimo, isso envolve o uso de técnicas e ferramentas de gestão de configuração e controlo de versão, e explorar as ligações de rastreabilidade para

monitorizar e controlar o impacto das mudanças em diferentes partes da documentação. As alterações típicas em especificações de requisitos incluem a adição ou exclusão de requisitos, e correção de erros. Os requisitos são adicionados em resposta à evolução das necessidades dos interessados, ou porque foram esquecidos numa análise inicial. Os requisitos são excluídos normalmente durante o desenvolvimento, para evitar custos desnecessários e cumprimento de calendário, uma prática conhecida como *scrubbing* [Boe91]. Em qualquer caso, a gestão de inconsistências [GN99], em especificações de requisitos à medida que evoluem é um grande desafio, elas surgem como resultado de erros, e também por causa de conflitos entre os requisitos existentes. Cada inconsistência implica que é necessário alguma ação para identificar a causa e procurar uma solução [HN95].

A gestão das mudanças nos requisitos não é apenas um processo de gestão da documentação, é também um processo de reconhecimento de mudança através da continuidade de elicitação de requisitos, reavaliação de risco e avaliação de sistemas no seu ambiente operacional. Em engenharia de software, tem sido demonstrado que focar a mudança no código do programa leva a uma perda de estrutura e sustentabilidade [BR00]. Assim, cada mudança proposta precisa de ser avaliada em termos de requisitos e arquitetura existente para que a compensação entre o custo e o benefício de fazer uma mudança possa ser avaliada.

2.3 Classificação de Requisitos

Como referido anteriormente os requisitos de software podem ter vários tipos. De um modo muito generalista, e no âmbito desta dissertação, são identificadas duas categorias de requisitos principais: Funcionais e Não Funcionais. Segue-se um breve explicação das mesmas.

2.3.1 Requisitos Funcionais

Os requisitos funcionais podem assumir a forma de cálculos, detalhes técnicos, manipulação e processamento de dados entre outras funcionalidades específicas que definem o que um sistema deve realizar. Estes requisitos são definidos principalmente olhando para casos de uso específicos. Os requisitos funcionais são suportados por requisitos não-funcionais (também conhecidos como requisitos de qualidade), que impõem restrições sobre a implementação. Requisitos funcionais são definidos como especificações de serviços e/ou métodos que o sistema a que se referem deve fornecer, como deve reagir em situações específicas e como se comportar em determinadas ações [Som10]. Geralmente, os requisitos funcionais são expressos na forma de "sistema deve fazer <requisito>". Seguem alguns exemplos:

- "The user shall be able to search either all of the initial set of databases or select a subset from it."
- "The system shall provide appropriate viewers for the user to read documents in the document store."
- "The authentication mechanism shall control the user's access"

Um requisito funcional contem um nome e número único, uma breve descrição e o porquê de ele existir. Estas informações são usadas para ajudar o leitor a entender por que o requisito é

necessário, e para acompanhar a evolução do mesmo ao longo do processo de desenvolvimento do sistema [Som10].

2.3.2 Requisitos Não Funcionais

O outro tipo de requisitos trabalhado são os requisitos não funcionais. Estes definem propriedades e restrições do sistema, por exemplo, tempo de resposta e requisitos de armazenamento. As restrições aplicam-se às capacidades de E/S do dispositivo, representações do sistema, etc. Na Figura 2.1 é apresentado um esquema das subdivisões possíveis dentro dos requisitos não funcionais.

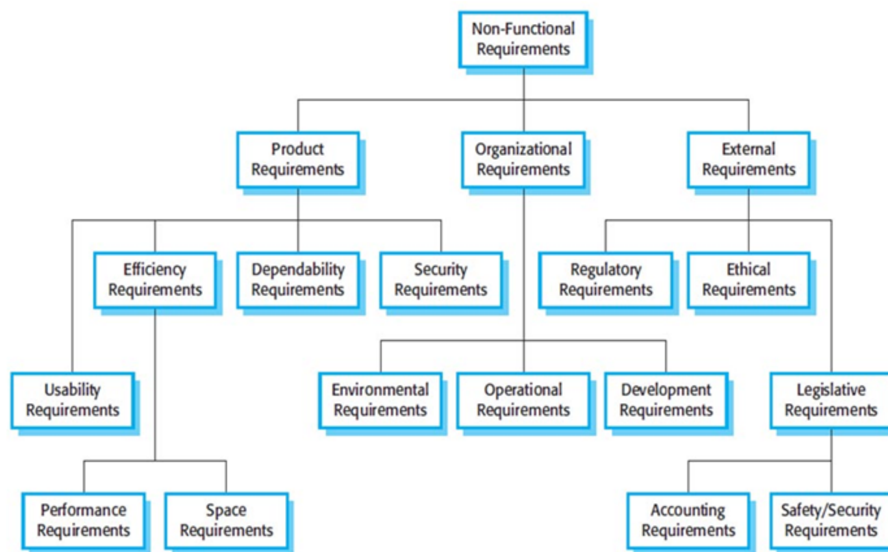

Figura 2.1: Esquema das subdivisões dos Requisitos Não Funcionais

Devido à grande quantidade de requisitos não funcionais existente, e dado o objetivo deste projeto, houve a necessidade de encurtar o número de categorias. Segundo [DWT12] podemos agrupar os requisitos não funcionais em quatro tipos principais:

- "Operacional"
- "Performance"
- "Security"
- "Cultural and Political"

No entanto, estas categorias não são suficientes para sistemas de grande dimensão, que apresentam uma maior complexidade e proporcionalmente uma maior quantidade de requisitos. Tendo em conta o objetivo final da classificação que foi desenvolvida no âmbito do projeto propõem-se que os classificadores devolvam como resultado final as seguintes categorias:

1. *Functional*: Descrito na Secção 2.3.1;
2. *Usability*: Descrevem requisitos que se refiram à *user experience*, normalmente referem-se a tempos de espera, normas de navegação e ajudas, são facilmente confundidos com requisitos funcionais;

3. *Efficiency*: Referem principalmente requisitos de hardware que é necessário para que o software funcione normalmente;

4. *Dependability*: Este tipo de requisitos descrevem restrições relativas às falhas que os componentes e/ou aplicação permitem;

5. *Security*: Requisitos associados aos métodos de segurança e recuperação que o sistema deve incluir, por exemplo, autenticações, intrusões, backups, recuperações;

6. *Operational*: Descrevem requisitos que especificam capacidades do sistema que garantem o estável funcionamento do sistema. São muitas vezes mencionados em questões de preservação e armazenamento;

2.4 Principais Desafios

A ER é um processo fundamental para o ciclo de vida do desenvolvimento de software. Alguns erros nos requisitos não são identificados durante o desenvolvimento, permanecendo escondidos até que o sistema se torne operacional obrigando a que as necessidades dos clientes não sejam cumpridas. Requisitos mal definidos não só levam a modificações nas especificações dos requisitos, como exigem nova projeção, implementação e teste de todo o software. Por isso, os engenheiros de requisitos têm que lutar contra um grande número de desafios para um desenvolvimento de software eficaz e eficiente. Antecipar os desafios de ER vai fornecer oportunidades para engenheiros de requisitos melhorarem a taxa de sucesso de software. Desafios de engenharia de requisitos foram classificados em componentes [AU10]:

- Tecnológicos;
- Processo de ER;
- Organizacionais;
- Conflitos dos interessados;
- Tempo.

Os desafios tecnológicos estão maioritariamente relacionados com o uso de novas tecnologias e seu constante avanço, que pode constituir um entrave a uma boa especificação de requisitos. Requisitos definidos aquando o uso de uma tecnologia podem não ser satisfeitos aquando a migração para uma nova ou atualização da mesma. É necessário um bom conhecimento das ferramentas nas quais os requisitos foram definidos para saber se os impactos da mudança serão ou não benéficos e se se verifica o cumprimento dos requisitos depois da mesma ser efetuada.

O objetivo do processo de ER é investigar quais as tarefas que precisam de ser realizadas e quais são os limites e restrições em software. Aquisição e compreensão dos requisitos para domínios complexos ou sistemas críticos sempre foram grandes desafios para os engenheiros de requisitos. Além disso, os interessados não expressam os seus requisitos corretamente durante o processo de elicitação. Como resultado, as especificações de requisitos são vagas e ambíguas. A validação de requisitos melhora a probabilidade de sucesso do projeto, uma vez validados, são desenvolvidos os protótipos para garantir que os requisitos e solução são corretos. No entanto, o protótipo pode fornecer detalhes insuficientes devido à ocorrência de erros e a correção desses

erros podem levar a atrasos.

As aplicações de software são desenvolvidas com a colaboração de estratégias de negócios. No entanto, infelizmente, existe uma extensa diversidade de perceções dentro dos departamentos organizacionais. Assim, o alinhamento e sincronização estratégicas defendidas por diferentes departamentos tornou-se uma tarefa crítica para os engenheiros de requisitos. Adicionalmente, um processo de negócio eficaz representa um funcionamento eficiente de uma organização. Apesar disso, as organizações estão-se a concentrar rapidamente em redesenhar processos de negócio para fazer mudanças substanciais e melhorias no seu nível de desempenho. Eventualmente, mudanças no processo de negócio podem também transformar os requisitos de software, adquirindo assim um esforço substancial na gestão de requisitos voláteis, que estabelecem grandes desafios à ER.

Um projeto bem sucedido tem uma grande influência sobre os interessados, caso contrário, o software pode enfrentar riscos significativos [Tur08]. Habilidades técnicas inadequadas dos engenheiros de requisitos e a falta de conhecimentos dentro de um domínio podem ter um grande impacto no software. Engenheiros de requisitos são incapazes de tratar adequadamente os problemas e as necessidades do utilizador final. Além de que, alguns engenheiros de requisitos experientes podem ser ignorantes no que diz respeito a ferramentas de ER emergentes. Este tipo de ineficiência pode levar a várias perceções do que os interessados realmente precisam e, consequentemente, a uma má elicitação de requisitos que leva ao desenvolvimento de um software que não corresponde às necessidades de quem o encomendou.

O agendamento é um processo para o planeamento e gestão do tempo. O agendamento é uma das tarefas predominantemente difícil e totalmente fundamental para o sucesso do software. No entanto, geralmente o tempo necessário na realização de tarefas durante a fase de ER é subestimado. Como resultado, a finalização de tarefas fica adiada particularmente quando as tarefas estão num caminho crítico. Estes desafios requerem que os engenheiros de requisitos giram e realizem tarefas ilimitadas, levando-os muitas vezes a seguir atalhos ou ignorar o foque de aspetos importantes. Consequentemente, os requisitos são mal estabelecidos e o projeto fica atrasado.

2.5 Soluções Existentes

Com base num estudo realizado [BS09] apresenta-se a tabela comparativa das principais ferramentas usadas na ER.

Observando a Tabela 2.1 todas as ferramentas apresentadas oferecem a possibilidade de TRS (*Technical Requirements Specification*), ou seja, construção do documento que contém os requisitos de determinado software, e oferecem também uma interface externa que permite integração com outros softwares usados na ER. Na sua maioria as ferramentas de ER apresentadas já vêm com modelos predefinidos de requisitos que o engenheiro de requisitos pode optar por seguir ou começar em branco assim como permitem a prototipagem e auditorias a qualquer um dos requisitos. Em termos de escalabilidade e glossários as ferramentas estão um pouco limitadas pois não são muitas as que fornecem essas capacidades. (Ver Glossário para definições e mais informação).

Tools \ Attributes	Glossary & Ontology	Checklist	Templates	Use Case Modelling	Prototyping & Audit	TRS	Scalability	External Interface
Requisite Pro	x	x	√	√	√	√	x	√
Case Complete	√	x	√	√	√	√	x	√
Analyst Pro	x	x	x	√	x	√	√	√
Optimal Trace	x	x	√	√	√	√	x	√
DOORS	x	x	√	x	√	√	√	√
GMARC	x	x	√	x	√	√	x	√
Objectiver	√	√	√	x	√	√	x	√
RDT	√	√	√	x	√	√	√	√
TcSE	x	x	x	√	x	√	x	√
Code Assure	x	x	x	x	x	√	x	√

Tabela 2.1: Tabela comparativa das ferramentas usadas na ER.

2.6 Métodos Desenvolvidos

Nesta dissertação foram desenvolvidos dois classificadores de requisitos de software que podem ser inseridos numa ferramenta já desenvolvida para gestão e armazenamento de requisitos ou então como uma nova ferramenta. Os dois classificadores funcionam de maneira distinta uma vez que têm como base diferentes áreas da engenharia informática. Um classificador baseia-se em métodos formais e PNL, enquanto que o outro é baseado em métodos estatísticos. Ambos recebem o mesmo tipo de dados de entrada, texto pertencente a requisitos de software em língua inglesa, por exemplo, "*The system must allow connections to other modules.*".

Além dos classificadores foi criado um método para atribuir etiquetas sintáticas a texto. Estas etiquetas definem a classe sintática a que a palavra pertence. Essa atribuição é feita assim que o classificador recebe os dados de entrada, transformando o texto normal, numa frase do género "*The/DT system/NN must/MD allow/VB connections/NNS to/TO other/JJ modules/NNS ./.*"de modo a ajudar na classificação do requisito.

Depois do texto do requisito etiquetado são efetuados os procedimentos, definidos em detalhe no Capítulo 3, para cada um dos classificadores retornando no final do processo a classificação individual de cada requisito de software.

2.7 Conclusões

Em termo de conclusão, pretendeu-se com esta secção dar a conhecer o processo de ER destacando a complexidade de todo o processo, assim como as principais atividades pelas quais ela passa, e ainda como as técnicas por elas usadas. Pretendeu-se também dar a conhecer as definições de requisito de software, assim como os vários tipos de requisitos, funcionais e não funcionais, e as discordâncias entre a definição destes últimos, explicando de seguida a forma como os quais foram tratados ao longo desta dissertação. Os desafios que afetam o processo de ER foram descritos também para dar a ideia dos problemas que os engenheiros de requisitos enfrentam no seu dia a dia. O principal objetivo da análise das ferramentas existentes utilizadas no processo de ER tem a ver com a novidade que esta dissertação trás, pois nenhuma das principais ferramentas têm qualquer tipo de classificação de requisitos.

Capítulo 3

Implementação dos Algoritmos

3.1 Introdução

Neste capítulo é feita uma descrição dos dois classificadores implementados, são apresentadas as fases da implementação destes classificadores, em *.NET Framework* especificamente em *C#*, bem como os passos necessários para a utilização dos mesmos. É apresentada também uma comparação entre as tecnologias estudadas para a criação dos classificadores aqui descritos.

Tendo em conta os pouco exemplos de requisitos que foi possível recolher para trabalhar, os classificadores foram divididos consoante o tipo de requisitos que classificam, isto é, o classificador baseado em métodos formais e PLN classifica apenas requisitos funcionais, e o classificador baseado em métodos estatísticos classifica apenas requisitos não funcionais, especificamente requisitos de *security* e *dependability*. De notar que apesar desta divisão de tipos de classificação, ambos os classificadores podem ser adaptados para classificarem qualquer tipo de requisitos, funcionais ou não funcionais.

3.2 POS Tagger

Um *Part-Of-Speech Tagger* (*POS Tagger*) é um software que recebe como entrada um texto numa linguagem humana e atribui etiquetas correspondentes a partes do discurso para cada palavra (e outros sinais de pontuação), como substantivos, verbos, adjetivos, etc. A maioria dos sistemas de etiquetagem são baseados numa estrutura gramatical livre de contexto. Dentro desta base, é importante distinguir entre a *Skeleton Parsing* e *Full Parsing*. *Full parsing* visa proporcionar o mais detalhado possível uma análise da estrutura sintática da frase, enquanto que *Skeleton Parsing*, tem uma abordagem menos detalhada que tende a utilizar um conjunto de etiquetas sintáticas menos distintas e ignora, por exemplo, a estrutura interna de certos tipos de constituintes [Nor14]. Note-se o exemplo:

(S(N Nemo/NP1 ,/, (N the/AT killer/NN1 whale/NN1 N) ,/, (Fr(N who/PNQS N)(V ́d/VHD grown/VVN (J too/RG big/JJ (P for/IF (N his/APP$ pool/NN1 (P on/II (N Clacton/NP1 Pier/NNL1 N)P)N)P)J)V)Fr)N) ,/, (V has/VHZ arrived/VVN safely/RR (P at/II (N his/APP$ new/JJ home/NN1 (P in/II (N Windsor/NP1 (safari/NN1 park/NNL1)N)P)N)P)V) ./. S)

O exemplo mostra um *Full Parsing*, onde a estrutura sintática é indicada por pares (*S*, *N*, *P*, *V*) definindo frase, sintagma nominal, predicado e verbo respetivamente, e as palavras têm etiquetas que identificam o tipo de função gramatical que lhe é inerente, como adjetivo (*JJ*) ou verbo no pretério perfeito (Vn). De seguida apresenta-se um exemplo de *Skeleton Parsing*:

(S (P For/IF (N the/AT members/NN2 (P of/IO (N this/DD1 university/NNL1 N)P)N)P) (N this/DD1 charter/NN1 N) (V enshrines/VVZ (N a/AT1 victorious/JJ principle/NN1 N)V)S) ;/; and/CC (S(N the/AT fruits/NN2 (P o/IO (N that/DD1 victory/NN1 N)P)N) (V can/VM

immediately/RR be/VB0 seen/VVN (P in/II (N the/AT international/JJ community/NNJ (P of/IO (N scholars/NN2 N)P) (Fr that/CST (V has/VHZ graduated/VVN here/RL today/RT V)Fr)N)P)V)S./.

No *Skeleton Parsing* a estrutura sintática varia. Todos os sintagmas nominais são simplesmente etiquetados com a letra *N*, enquanto que no exemplo de *Full Parsing*, existem vários tipos de sintagma nominal que se distinguem de acordo com características como, por exemplo, a pluralidade (*Ncs*) [Shi09].

No âmbito desta dissertação, foi usado como ferramenta de etiquetagem o *POS* feito na Universidade de *Stanford*, o *Stanford POS Tagger* na versão 3.3.1 [TKMS14], onde as etiquetas sintáticas usadas são mais refinadas, por exemplo, *substantivo plural*, ou *verbo no infinitivo*, ou *verbo no pretérito perfeito*.

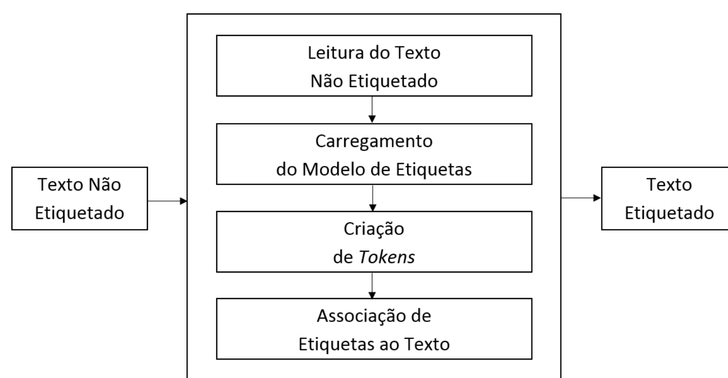

Figura 3.1: Esquemática do funcionamento do *Stanford POS Tagger*

O funcionamento do *Stanford POS Tagger* tem de começar obrigatoriamente com o fornecimento de um conjunto de texto não etiquetado, de notar que o *Stanford POS Tagger* tem também a opção de etiquetar texto recebendo um ficheiro com texto. Ao receber o texto o *POS Tagger* terá de carregar o texto para memória seguido do modelo definido no âmbito do desenvolvimento do *Stanford POS Tagger* para a língua inglesa. Para proceder à atribuição de etiquetas às frases é necessário criar *tokens* (cadeia de um ou mais caracteres que tem importância enquanto grupo) através da *tokenization* (processo de criação de *toknes* a partir de um sequência de entrada de caracteres). Uma vez estes componentes todos obtidos, são associadas as etiquetas aos *tokens* conforme o modelo previamente carregado. No final das etiquetas atribuídas é retornada uma cadeia de texto, já etiquetados com as etiquetas *POS*.

As etiquetas sintáticas usadas no âmbito desta dissertação, são as etiquetas criadas e usadas pelo projeto *Penn Treebank*, descritas na Tabela 3.1. O projeto *Penn Treebank* anota texto em linguagem natural numa estrutura linguística. Mais recentemente, foram feitos *skeleton parsers* que mostram informações sintáticas e semânticas. As anotações de texto acima referidas são feitas também com as etiquetas *POS* [MTM90].

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Tabela 3.1: Tabela com a listagem das etiquetas POS usadas no Penn Treebank

3.2.1 Implementação

Nesta dissertação foi usado um *POS Tagger*, baseado no *Stanford POS Tagger*. Sendo o mesmo feito em linguagem JAVA, foi necessário fazer alterações para que o mesmo pudesse ser usado no âmbito deste projeto, nomeadamente fazer a sua portabilidade para a *.NET Framework*, mais especificamente *C#*.

A utilização do *Stanford POS Tagger* na *.NET Framework* foi possível devido ao trabalho de Sergey Tihon [Tih13b], que inicialmente fez a portabilidade do código JAVA do *Stanford POS Tagger* para usar com a linguagem *F#* [Tih13a] usando a tecnologia *IKVM.NET* [Fri11], e que depois converteu também para outra linguagem *.NET*, *C#* [Tih14]. A tecnologia *IKVM.NET* é uma *Java Virtual Machine* (JVM) para execução em *.NET* e *Mono*. Numa altura em que a maioria das pessoas na indústria de computadores considera que *JAVA* e *.NET* são tecnologias mutuamente

exclusivas, a *IKVM.NET* assume a responsabilidade de as reunir. O projeto nasce inicialmente da frustração com as limitações das ferramentas. Criada por Jeroen Frijters, ele propôs a criação de uma maneira de migrar um aplicativo de *JAVA* existente para *.NET* [Fri11].
Para a utilização do *POS Tagger*, no âmbito desta dissertação, foi necessário criar uma classe em *C#* e testar o seu desempenho numa aplicação dedicada exclusivamente a testes. A classe criada define duas funções principais:

- uma função para atribuição de etiquetas;
- função auxiliar para retorno do texto etiquetado.

A função para atribuição de etiquetas pode ser dividida em duas partes: (i) carregamento do modelo, onde o modelo de classificação é carregado; (ii) o *POS Tagger* recebe o texto por parâmetro e em conjunto com o modelo carregado atribui etiquetas a cada pedaço de texto, retornando no final o texto etiquetado em forma de lista.

Consideremos a seguinte frase referente a um requisito funcional:

"The system should produce a warning when the temperature reaches 90."

Depois de a mesma frase ser passada ao *POS Tagger*, o retorno-no feito por ele será a mesma frase etiquetada de acordo com o modelo carregado previamente.

"The/DT system/NN should/MD produce/VB a/DT warning/NN when/WRB the/DT temperature/NN reaches/VBZ 90/CD ./."

O uso deste tipo de etiquetagem, surge da necessidade de uma pré classificação sintática antes de qualquer tipo de classificação dos requisitos em categorias. O facto de cada palavra do requisito ter associada uma etiqueta sintática conforme a sua propriedade frásica ajuda a um processamento por parte dos métodos de classificação baseados em linguagens formais, uma vez que atribuí mais informação aos símbolos definidos na gramática.

3.3 Classificação de Requisitos

O classificador é o elemento principal desta dissertação, uma vez que o objetivo da mesma é a criação de métodos para classificação automática de requisitos de software. Foram criados dois classificadores distintos: um classificador baseado em métodos formais de análise de texto com conceitos de PLN, e um classificador baseado em métodos estatísticos. Os mesmos vão ser descritos em detalhe nas secções seguintes.

3.3.1 Classificador Baseado em Métodos Formais e PLN

Esta secção da dissertação descreverá um classificador de texto baseado em métodos formais e PLN. Este classificador pode ser divido em duas partes distintas: gramática e *parser*.

A fim de analisar dados em linguagem natural com métodos formais, deve-se primariamente acordar a gramática a ser utilizada pelo *parser*. A escolha da sintaxe é afetada pelas preocupações linguísticas e computacionais, por exemplo, alguns sistemas de análise usam uma gramática lexical funcional (modelo de gramática que fornece estruturas para examinar estruturas morfológicas e estruturas sintáticas de uma frase ou texto [Nor]).

Uma gramática é um conjunto de regras de sequências de texto numa linguagem formal. As regras descrevem como formar cadeias do alfabeto da língua que são válidas de acordo com a sintaxe da linguagem. A gramática não descreve o significado das cadeias de texto ou o que pode ser feito com elas, descreve apenas a sua forma.

No âmbito desta dissertação é fundamental definir também o conceito de gramática formal. Define-se gramática formal como um conjunto de regras para reescrita de sequências de texto, juntamente com um *start symbol* para identificação de onde a reescrita começa. No entanto, e sendo o seu principal uso nesta dissertação, este tipo de gramáticas, também podem ser usadas como base para uma função de reconhecimento em computação, com o objetivo de determinar se uma dada cadeia de caracteres pertence a uma linguagem ou se é gramaticalmente incorreta.

A gramática elaborada no âmbito desta dissertação é uma gramática formal, definida de modo a identificar estruturas de requisitos funcionais. Foi construída com base em exemplos de requisitos funcionais recolhidos e moldada, inicialmente de acordo com os primeiros exemplos recolhidos. A mesma gramática, sofreu bastantes alterações durante o desenvolvimento, evoluindo em conteúdo, não só consoante os testes efetuados mas também, com requisitos funcionais que foram recolhidos já depois da primeira versão da gramática, ajudando ao refinamento da mesma. A gramática, na sua forma final, pode ser consultada no Anexo A.1.

Para descrever a função da gramática assumimos que o seguinte texto de um requisito funcional foi etiquetado pelo *POS Tagger* e será agora exposto às regras da gramática:

"The/DT system/NN must/MD accept/VB new/JJ users/NNS ./."

Para que este requisito seja classificado corretamente como um requisito funcional, a gramática terá de conter uma regra onde os símbolos que este requisito possuí estejam descriminados e inseridos numa estrutura válida gramaticalmente, isto é, a gramática tem de conter uma regra que diga que a estrutura frásica do texto do requisito pertence ao grupo dos requisitos funcionais, essa regra é descrita de seguida:

```
funcional.Rule = funcionalBase | funcionalAdv | funcionalNext;
funcionalBase.Rule = determinante + nomes + modal + accao + (unidadePontuac | Empty);
```

O texto de requisito terá de ser inserido na regra *funcionalBase.Rule* para que seja aceite pela gramática desenvolvida. Para que isso aconteça o mesmo requisito terá ter uma estrutura que se insira no definido pelas variáveis *determinante*, *nomes*, *modal*, *accao* e respeitar a condição de fim. Como o próprio nome indica os primeiros três componentes da regra *funcionalBase* estão à espera de uma determinante, um ou mais nomes e um verbo modal. A composição *accao* define-se na regra:

```
accao.Rule = complementoDireto |
      (verbo + adjetivo) + nomes |
      verbo + adjetivoTo + nomes |
      (verbo + adjetivo) + complementoDireto |
      verbo + adjetivoTo + complementoDireto |
      (verbo + adjetivo + preposicao) + complementoDireto |
      verbo + adjetivoTo + preposicao + complementoDireto
      ;
```

No caso da frase *"The/DT system/NN must/MD accept/VB new/JJ users/NNS ./."* a parte *"accept/VB new/JJ users/NNS ./."* será captada pela componente da *accao*, mais especificamente pela linha *(verbo + adjetivo) + nomes*. Depois de esta estrutura estar definida, será função do

parser criar uma *parse tree* e atribuir uma classificação ao requisito.

Convém clarificar algumas definições relativas ao termo *parser* e ao processo de *parsing*. *Parsing* define-se como o processo de analisar uma sequência de símbolos, tanto em linguagem natural ou em linguagens de programação, de acordo com as regras de uma gramática formal definida.

O termo tem significados ligeiramente diferentes conforme os ramos da linguística computacional ou engenharia informática. O *parsing* tradicional de uma frase é frequentemente usado como um método de compreensão do significado exato de uma frase, realçando a importância das divisões gramaticais como o sujeito e predicado.

Dentro de linguística computacional, o termo é usado para se referir à análise formal através de um computador de uma frase ou outras sequências de palavras nos seus constituintes, resultando numa árvore de análise sintática, mostrando as relações entre constituintes, que também podem conter informações semânticas.

Dentro da engenharia informática, o termo é usado na análise das linguagens de programação, referindo-se à análise sintática do código de entrada, nas suas partes constituintes a fim de facilitar a escrita de compiladores e interpretadores.

A atividade gramatical tradicional do *parsing*, envolve partir um texto, ou frase, nas suas componentes do discurso com uma explicação sobre a forma, função e relação sintática de cada componente, isto é determinado em grande parte pelo estudo das línguas, conjugações e declinações, que podem ser bastante complexas para as línguas fortemente flexionadas (línguas onde a modificação de uma palavra pode expressar diferentes categorias gramaticais, como modo, tempo, pessoa, número e género [BB10]).

O foco desta dissertação implica um sistema de processamento de linguagem natural, isto é, textos escritos em linguagem humana (inglês) que irão ser analisados por um computador. Frases em linguagem natural tornam-se difíceis de ser analisadas por programas computacionais devido a uma ambiguidade substancial na estrutura da linguagem humana, cujo uso têm objetivo de transmitir um significado entre um intervalo de hipóteses quase ilimitado, mas só uma pequena porção desse intervalo é que se torna pertinente consoante o caso particular que se está a analisar.

A maioria dos *parsers* modernos são na maior parte estatísticos, ou seja, eles dependem de um conjunto de informação de treino, a partir de texto manualmente etiquetado. Esta abordagem permite ao sistema reunir informações sobre a frequência com que várias construções ocorrem em contextos específicos. A maioria dos sistemas mais bem sucedidos usam estatísticas lexicais (isto é, se consideram as identidades das palavras envolvidas, bem como a sua parte do discurso). No entanto, tais sistemas são vulneráveis a *overfitting* (ocorre quando um algoritmo ou modelo de aprendizagem automática estatística ajusta demasiado os dados ao conjunto de treino [Cai14]) e requerem algum tipo de cuidados para serem eficazes. O código do *parser* desenvolvido no âmbito desta dissertação pode ser consultado no Anexo A.2.

```
Parser p = new Parser(grammar);
ParseTree tree;
```

A classe acima representada mostra duas partes importantes do classificador baseado em métodos formais e PLN. a função *new Parser(grammar)* carrega a gramática que contêm as especificações da linguagem que se quer classificar e a variável *ParseTree* serve como estrutura de estração de informações para obter o resultado da classificação.

```
foreach (var result in tree.Root.ChildNodes)
{
  outputRequirements.Add(new Result(item, result.Term.ToString(), null));
}
```

A lista referida acima (*outputRequirements*) contém uma lista de estruturas, com três componentes: o texto etiquetado pertencente a um possível requisito funcional, a classificação do mesmo e uma possível mensagem de erro (caso ocorra).

Para a implementação do classificador baseado em métodos formais e PLN foram estudadas e usadas duas ferramentas de construção de *parsers*:

- Gardens Point LEX e Gardens Point Parser Generator;
- Irony - .NET Language Implementation Kit.

O *Gardens Point LEX* (GPLEX) [Gou14] é um gerador de *scanners* (programas que reconheceram padrões lexicais no texto) que produz *scanners* lexicais escritos em C#. A linguagem de entrada é semelhante à linguagem original de especificação do LEX [LMB95]. O GPLEX gera *scanners* baseados em torno de autómatos finitos. Os autómatos gerados têm um número de estados mínimo por definição. Os *scanners* gerados são projetados para interagir corretamente com *parsers* gerados pelo *Gardens Point Parser Generator* (GPPG), podendo ser usados por *parsers* gerados pelo *COCO/R* (gerador de compiladores, que usa uma gramática, atribuída de uma linguagem fonte e gera um scanner e um *parser* para essa linguagem).

O complemento do GPLEX é o *Gardens Point Parser Generator* (GPPG) é um gerador de *parsers* que produz *parsers* escritos em C#. A linguagem de entrada é igual à do *YACC* (ferramenta generalista para descrição dos dados de entrada de um programa de computador. O YACC especifica as estruturas de entrada, juntamente com o código a ser invocado como cada tipo de estrutura é reconhecido) [LMB95], e os *parsers* são *LALR(1)* [Cha87], sendo projetado para trabalhar com GPLEX. Os *parsers* gerados seguem uma abordagem *LALR(1)*. Define-se como *LALR(1)* um *parser* que interpreta o texto numa direção sem voltar atrás, essa direção é tipicamente da esquerda para a direita, produzindo uma derivação invertida mais à direita fazendo uma análise de baixo para cima. Para evitar retrocessos, o é permitido verificar os símbolos de entrada seguintes, lookahead, **K** símbolos antes de decidir como interpretar os símbolos anteriores, normalmente, *K* é igual 1, e não é mencionado.

Os *parsers* gerados são projetados para interagir corretamente com *scanners* gerados pelo GPLEX, no entanto, podem ser utilizados com *scanners* gerados pelo *COCO/R*.

Uma característica particular da ferramenta é a geração, opcional, de um relatório *HTML* que permite uma navegação fácil do autómato finito que reconhece os prefixos viáveis da linguagem especificada. Esse relatório mostra os itens de produção, símbolos de verificação à frente e

ações para cada estado do autómato, com o objetivo simplificar consideravelmente o diagnóstico de conflitos gramaticais.

Estas ferramentas foram no entanto descartadas no decorrer do desenvolvimento da dissertação por algumas razões como o desempenho, sendo duas ferramentas estavam demasiado interligadas para uma futura mudança e a complexidade da gramática, apesar da ferramenta apresentar algumas ajudas na resolução de conflitos, exige uma sintaxe muito própria e oferece pouca documentação sobre as palavras chave com as quais a linguagem está à espera. Tendo em atenção todos estes fatores a ferramenta com a qual o classificador baseado em métodos formais e PLN foi desenvolvido foi a *Irony - .NET Language Implementation Kit*.

O *Irony* [Iva14] é um kit de desenvolvimento para a implementação de linguagens na *.NET Framework*. Ao contrário da maioria soluções das *LEX/YACC* existentes, o *Irony* não utiliza qualquer *scanner* ou *parser* para geração de código a partir de especificações gramaticais. Em *Irony* a gramática da linguagem é codificada diretamente em *C#* usando operadores para expressar construções gramaticais, os módulos de *scanner* e *parser* do *Irony* usam a gramática, codificada numa classe *C#*, para controlar o processo de análise.

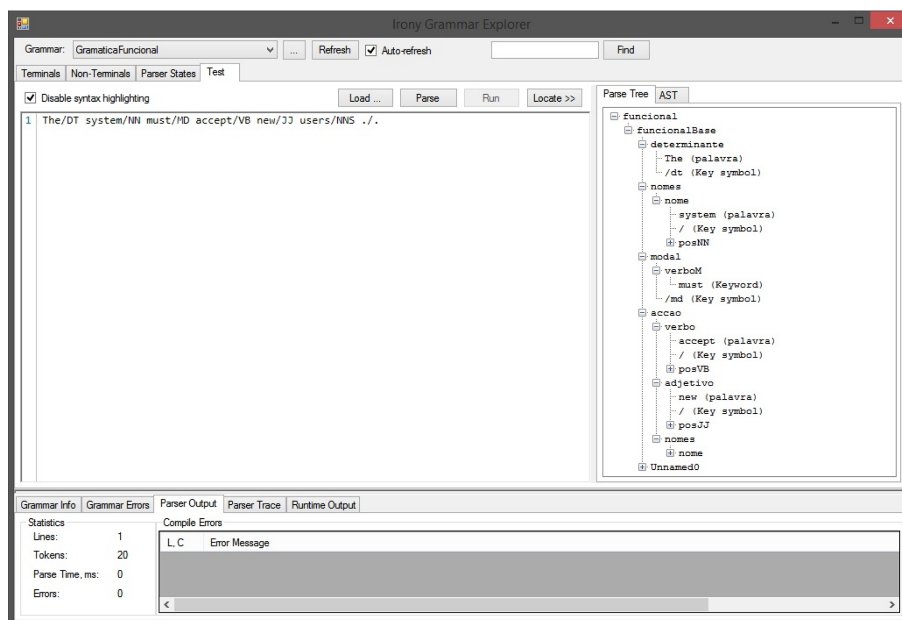

Figura 3.2: Janela principal do *Irony Grammar Explorer*

O *Irony Grammar Explorer* é uma ferramenta disponibilizada pelo *Irony* que ajuda na visualização e criação da gramática. Apesar da mesma ser escrita em *C#* é necessário criar uma biblioteca para ser lida pela ferramenta, permitindo executar tarefas como testes, resultados, componentes distintas da gramática definida.

A fase final do classificador baseado em métodos formais e PLN passa pela junção das duas componentes descritas acima, o *POS Tagger* e os métodos criados com o *Irony*, de modo a funcionar como mostra a Figura 3.3. De notar que os exemplos apresentados na gramática (Anexo A.1) e no *parser* (Anexo A.2) já estão ambos escritos com a linguagem *C#*, a gramática já está definida com funções da *Irony* e o *parser* usa componentes disponibilizadas pela *Irony* também.

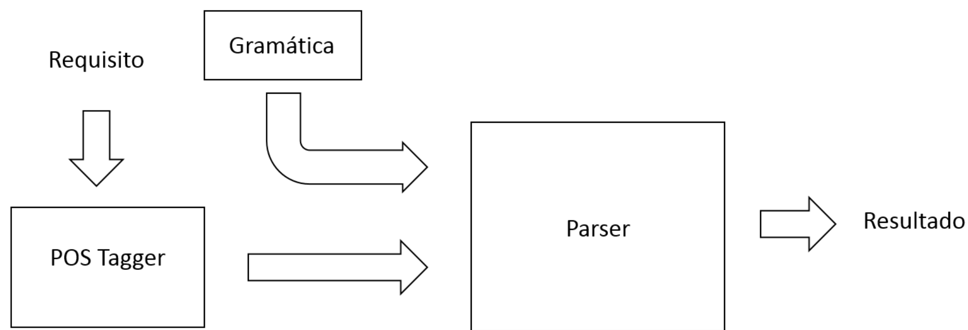

Figura 3.3: Funcionamento do Classificador Baseado em Métodos Formais e PLN

A Figura 3.3 descreve de uma maneira sucinta o funcionamento do classificador baseado em métodos formais e PLN. Tal como no *POS Tagger* tem obrigatoriamente que iniciar o processo recebendo algum tipo de texto, neste caso será um requisito funcional, que passará pelo *POS Tagger* e que devolve o mesmo já etiquetado com as etiquetas já referidas na Secção 3.2. Este texto etiquetado é então fornecido ao *parser* que depois da gramática ser carrega, usa as suas funcionalidades para dizer se o texto do requisito pertence ou não ao conjunto dos requisitos funcionais (Estas funcionalidades serão explicadas mais detalhadamente no Capítulo 4).

```
file:///C:/Users/Leopoldo/Desktop/Irony Stress Test/RPI/RPI/TestConsole/bin/Debug/TestConsole.EXE
Reading POS tagger model from wsj-0-18-bidirectional-nodistsim.tagger ... done [3,0 sec].
Recognised Requirements:
Result: funcionalBase - Req: All/DT clients/NNS should/MD be/VB listed/VBN by/IN the/DT system/NN ./.
Result: funcionalAdv - Req: When/WRB temperature/NN reaches/VBZ 90/CD the/DT system/NN should/MD produc
Result: funcionalBase - Req: The/DT system/NN should/MD be/VB able/JJ to/TO list/VB all/DT clients/NNS
Result: funcionalAdv - Req: The/DT system/NN should/MD produce/VB a/DT warning/NN when/WRB the/DT tempe
Result: funcionalBase - Req: The/DT system/NN must/MD accept/VB new/JJ users/NNS ./.
Result: funcionalNext - Req: New/JJ users/NNS must/MD be/VB created/VBN by/IN the/DT system/NN ./.
Result: funcionalAdv - Req: If/IN salary/NN is/VBZ over/IN 9000/CD $/$ the/DT system/NN should/MD alert
Result: funcionalAdv - Req: The/DT user/NN should/MD be/VB alerted/VBN if/IN salary/NN is/VBZ over/IN 9
Unrecognised Requirements:
```

Figura 3.4: Exemplo de classificação usando o Classificador Baseado em Métodos Formais e PLN

Como se pode ver pela Figura 3.4, o classificador reconheceu o texto fornecido como sendo frases pertencentes a requisitos funcionais, mostrando à esquerda o tipo de categoria do requisito e à direita o texto do requisito classificado.

3.3.2 Classificador Baseado em Métodos Estatísticos

O classificador baseado em métodos estatísticos implementado no âmbito desta dissertação baseia-se num método de calculo de valores de frequência que uma determinada lista de palavras previamente definidas ocorre no texto/frase que se quer classificar. O método classifica o texto do requisito recebido em duas categorias definidas.

A construção do classificador baseado em meios estatísticos, pode ser dividida em três partes distintas:

- extração de palavras relevantes;
- função de classificação;
- funções auxiliares;

O classificador desenvolvido tem uma forte dependência numa lista de termos relevantes previamente definidos. O método para obtenção das listas de palavras mais relevantes em requisitos não funcionais, é um método de classificação de relevância de termos numa região de um texto, usando a seguinte fórmula:

$$Relevance = \log_2 \frac{P(w|R)}{P(w)} \quad (3.1)$$

onde a relevância de uma palavra *w* é obtida através do logaritmo de base dois, da divisão da probabilidade da palavra *w* ocorrer na região *R*, pela probabilidade da palavra *w* ocorrer em todo o texto. Descrevendo um exemplo prático: consideremos um texto com 1000 (mil) palavras, e que a frequência de uma palavra *w1* no texto é 100 (cem) e uma palavra *w2* é 4 (quatro). Definindo uma região *R* de 100 (cem) palavras e sabendo que a palavra *w1* aparece 9 (nove) vezes e a palavra *w2* ocorre 3 (três) vezes, queremos saber qual a palavra, *w1* ou *w2* que tem maior relevância. Uma vez que na região *R* a palavra *w2* ocorre 3 (três) das 4 (quatro) vezes que ocorre num todo, a palavra *w2* terá uma maior relevância na região.

Uma vez que este tipo de métodos requer uma grande quantidade de exemplos para uma recolha eficaz de informação, e não havendo dados suficientes para uma recolha eficiente, as palavras mais relevantes escolhidas para a lista resultam de uma análise manual do conjunto de exemplos disponível.

A função principal referida é a responsável pela condução do processo de classificação. Será a função que é chamada de dentro da biblioteca criada para o sistema externo a fim de obter uma classificação para o texto do requisito fornecido. O classificador está preparado para receber uma conjunto de listas de palavras relevantes para diferentes categorias, em contexto de exemplo estão implementadas duas categorias de requisitos não funcionais, *security* e *dependability*.

As funções auxiliares têm aqui uma função de suporte a todo o processo e nunca poderão ser consideradas descartáveis ou de menor importância pois sem elas o processo de classificação não aconteceria. Elas estão encarregues da contagem de palavras, somatório de frequências, conversões necessárias entre tipos de estruturas de dados usados pelas funções e carregamento de ficheiros (com as listas de palavras mais frequentes).

O funcionamento do classificador é bastante simples e a sua implementação foi feita na linguagem *C#* e incorporado numa biblioteca. Como referido, o classificador tem uma função principal que está responsável pela sequência de todo o processo. Esta sequência define o encadeamento das diferentes fases do processo de classificação:

- carregamento das listas com as palavras mais relevantes;

- leitura do texto dos requisitos;
- obter frequências das palavras mais relevantes no requisito;
- fazer os cálculos necessários para obter as frequências da palavra na frase;
- comparação da maior frequência entre as duas listas para um requisito;
- apresentar a classificação.

O processo de implementação passa então numa primeira fase pelo carregamento dos ficheiros com as listas de palavras mais relevantes em requisitos não funcionais, seguido da leitura dos requisitos que irão ser classificados. Para obter a frequência das palavras presentes nas listas num requisito específico é necessário comparar cada palavra presente na lista, com cada palavra do texto referente ao requisito. Uma vez obtida essa frequência é necessário armazenar essa informação tendo em conta qual a palavra em questão e a frequência da mesma no texto. O pedaço de código seguinte mostra as funções usadas para obter frequências das palavras nas frases e a média dessas mesmas frequências numa frase.

```
public string process(string text)
{
  freqXonT(securityMCW, text, freqSec);
  freqXonT(dependabilityMCW, text, freqDep);

  double sumSecurity = sumFreqX(freqSec, wordCount(text));
  double sumDependability = sumFreqX(freqDep, wordCount(text));

  (...)
}
```

A classificação do requisito será atribuída de acordo com a seguinte Fórmula 3.3.2.

$$\sum O(X) = \frac{Ocorrencias\,da\,palavra\,X}{Total\,de\,palavras\,da\,frase} \quad (3.2)$$

A classificação ocorre depois da mesma frase referente a um requisito ser analisada, o valor que a Fórmula 3.3.2 retorna ($\sum O(X)$) será comparado consoante o número de listas existentes, no caso desta dissertação serão duas, e verá para qual delas ocorrem mais palavras presentes nas mesmas listas, ilustrado no código seguinte:

```
public string process(string text)
{
  (...)
  if (sumSecurity == sumDependability)
  {
    category = "undefined";
  }
  else
  {
    if (sumSecurity > sumDependability)
    {
      category = "security";
    }
    if (sumDependability > sumSecurity)
    {
      category = "dependability";
    }
  }

  return category;
}
```

No final deste processo é retornada uma classificação para cada vez que a função *process* é chamada, isto implica uma chamada da função para cada requisito que precisa de ser classificado.

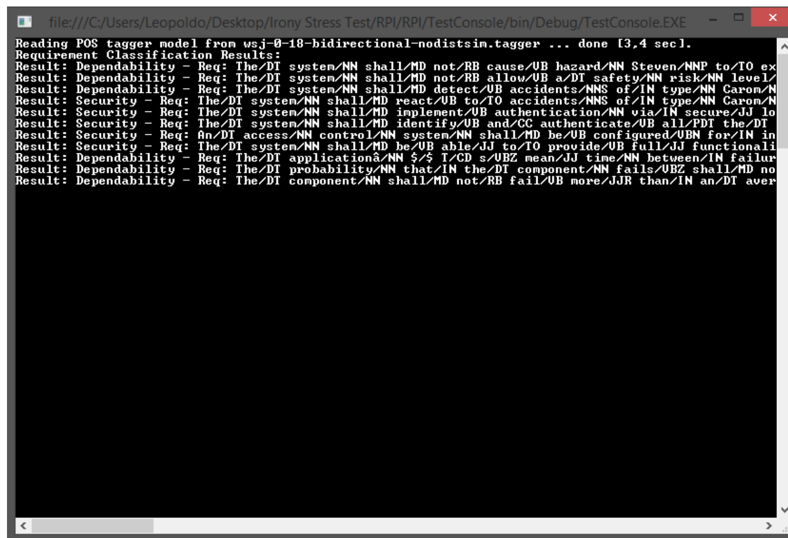

Figura 3.5: Exemplo de classificação usando o Classificador Baseado em Métodos Estatísticos

Como se pode ver pela Figura 3.5, o classificador reconheceu o texto fornecido como sendo frases pertencentes a requisitos não funcionais *security* e *dependability*, mostrando à esquerda o tipo de categoria do requisito não funcional e à direita o texto do mesmo requisito classificado.

3.4 Conclusões

Neste capítulo foi feita uma descrição dos dois classificadores implementados, são apresentadas as fases da implementação destes classificadores, em *.NET Framework* especificamente em *C#*, bem como os passos necessários para a utilização dos mesmos. É apresentada também uma descrições das tecnologias usadas para a criação dos classificadores aqui descritos.

Foram desenvolvidos dois métodos de classificação em *C#*, um baseado em métodos formais e PLN e outro com base em métodos estatísticos. Foi igualmente descrito e implementado um método para atribuição de etiquetas sintáticas a texto desenvolvido também em *C#*. Descreveu-se em detalhe a sua implementação e as decisões tomadas durante o seu desenvolvimento.

O objetivo deste capítulo prende-se à apresentação dos esquemas utilizados, das suas implementações e dos desafios e opções tomadas durante o desenvolvimento. Procura-se desta forma fornecer um visão aprofundada dos métodos e esquemas subjacentes utilizados na arquitetura especificada no capítulo seguinte.

Capítulo 4

Arquitetura

4.1 Introdução

No Capítulo 3 foram apresentados os diferentes componentes nos quais se baseia esta dissertação. Neste capítulo pretende-se abordar a estrutura da arquitetura, bem como os diferentes componentes que a compõem de uma forma pormenorizada. Pretende-se justificar as soluções implementadas e fornecer uma visão do funcionamento global, assim como de cada uma das fases.

4.2 Cenários de Utilização

A proposta inicial desta dissertação tinha como objetivo principal a criação de uma ferramenta de classificação de requisitos de software em linguagem natural, e que seria englobada no âmbito do projeto PROVA - *Platform for Software Verification and Validation*, levado a cabo pela empresa EDUCED.

O cenário de utilização, tendo em conta o objetivo principal desta dissertação, será o uso dos métodos apresentados no Capítulo 3 no interior de uma ferramenta de gestão de requisitos de software. O modo de utilização fica à responsabilidade do utilizador sendo que os métodos podem ser usados nos seguintes casos:

- como processo de classificação em *background*, onde o utilizador pode executar outras tarefas e deixar a classificação dos requisitos entregue a um dos métodos apresentados, apresentando depois o resultado quando concluído;
- como processo de classificação independente, onde o utilizador escreve todos os requisitos e no final chama o classificador, encarregando-o da classificação de um conjunto de requisitos, esperando o resultado;
- como processo de classificação em tempo real, onde o utilizador recebe a classificação do requisito assim que acaba a inserção do mesmo em sistema;
- como processo de classificação externo, onde o utilizador insere os requisitos em sistema, e o sistema proprietário é responsável pelo tratamento dos requisitos numa outra componente.

De notar que as ferramentas desenvolvidas no âmbito desta dissertação são para ser usadas exclusivamente pela EDUCED, no âmbito do projeto *PROVA* ou outros projetos da empresa.

4.3 Descrição da Biblioteca

Os métodos de classificação desenvolvidos e classes que os suportam estarão presentes numa *Dynamic-link library* ou *DLL*, sendo necessária a sua importação para o uso da mesma.

A biblioteca contêm no seu interior os seguintes itens:

- *Result*;
- *RPIException*;
- *GramaticaFuncional.cs*;
- *POSTagger.cs*;
- *Main.cs*;
- *Filter.cs*.

4.3.1 Classe *Result.cs*

A classe *Result.cs* é usada como estrutura de retorno na implementação do classificador baseado em métodos formais, uma vez que após a classificação do requisito é devolvida uma estrutura *Result* com três componentes distintas: o texto do requisito que foi analisado; a classificação obtida para esse requisito; e uma possível mensagem de erro que pode acontecer durante o processo de classificação. A classe pode ser vista no Anexo A.4.

4.3.2 Classe *RPIException.cs*

Na classe *RPIException.cs*, é definida uma estrutura para o tratamento de exceções. A classe permite o retorno discriminado de uma exceção uma vez que herda todas as propriedades da classe *Exception* [Net14] predefinida na linguagem *C#* e que permite não só aceder a muitas informações sobre a exceção que ocorreu, mas também permite enviar uma mensagem personalizada pelo método que a gerou.

4.3.3 Classe *GramaticaFuncional.cs*

Como referido anteriormente os classificador baseado em métodos formais e PLN tem uma componente gramatical bastante importante. Essa componente está definida na classe *GramaticaFuncional.cs* (Ver Anexo A.1). A classe define através da sintaxe da *Irony - .NET Language Implementation Kit* a gramática que o *parser* usado pelo classificador vai usar como suporte à classificação. É nesta classe que estão definidos os símbolos terminais e não terminais, assim como o símbolo inicial e os vários caminhos que a estrutura do texto do requisito pode formar de modo a que ele seja classificado como pertencente a uma categoria. A gramática descrita inteiramente nesta classe foi obtida através da análise de exemplos de requisitos funcionais recolhidos em várias pesquisas.

4.3.4 Classe *POSTagger.cs*

O *POS Tagger* definido no Capítulo 3 tem a sua implementação na classe *POSTagger.cs*. É aqui que estão definidas as funções que fazem a atribuição de etiquetas sintáticas ao texto de um requisito.

4.3.5 Classe *Main.cs*

A definição da *Main.cs* (ver Anexo A.2) é exclusiva ao classificador baseado em métodos formais e PLN. Nesta classe são chamadas as funções que permitem o *parsing* de cadeias de texto da *Irony - .NET Language Implementation Kit*. O pedaço de código abaixo mostra as três componentes importadas da biblioteca da *Irony*.

```
Parser p = new Parser(grammar);
ParseTree tree;
```

A função mais importantes é a *Parser* e a variável do tipo *ParseTree* merece também atenção.
A função *Parser* é a função que tem a responsabilidade de receber a gramática, uma vez que esta tem de ser passada por parâmetro, tornando o sistema mais escalável e a gramática mais dinâmica uma vez que pode ser alterada externamente à biblioteca, e passada mais tarde para dentro da mesma.
Quanto à variável *ParseTree* é nesta variável que acontece a classificação. A gramática foi concebida de modo a que a sua raiz, tenha informação suficiente de modo a que o classificador ao ver a estrutura da variável *ParseTree*, consiga classificar o texto do requisito numa categoria distinta.

4.3.6 Classe *Filter.cs*

O classificador baseado em métodos estatísticos assim como todos as funções que auxiliam o seu funcionamento estão definidas na classe *Filter.cs*.

```
public Filter(List<string> d, List<string> s)
{
  (...)

  dependabilityMCW = d;
  securityMCW = s;

  (...)
}
```

O processo inicia-se, obrigatoriamente, com o carregamento das listas dos termos mais relevantes nos tipos de requisitos presentes no texto que pretendemos classificar (*security* e *dependability* neste caso concreto). O excerto de código acima representa o início do processo de classificação onde as duas listas são recebidas por parâmetro e passadas para uma estrutura interna através da função *fillList*.

Uma vez carregadas as listas, é necessário proceder à classificação propriamente dita, com a verificação de quais e quantas vezes as palavras presentes nas listas ocorrem no texto que se quer classificar. Como mostra o pedaço de código seguinte:

```
private Dictionary<string, int> freqXonT(List<string> X, string t)
{
  (...)
  foreach (var word in X)
  {
    for(int i=0; i<text.Length; i++)
    {
      if (word == text[i].ToString())
      {
```

```
        count++;
      }
    }
    freqAux.Add(word, count);
    count = 0;
  }

  return freqAux;
}
```

As palavras mais relevantes presentes na lista assim como a frequência que elas ocorreram na frase são guardadas numa estrutura dicionário (*Dictionary*) interna que é passado à função responsável pelos cálculos.

```
private double sumFreqX(Dictionary<string, int> frequency, double length)
{
  (...)

  foreach (var pair in frequency)
  {
    sum = sum + Convert.ToInt32(pair.Value);
  }

  return ((double)sum/length);
}
```

O valor que é retornado por esta função é o quociente entre a soma total das palavras ocorridas e o total de palavras da frase que se analisou, e que determinará em qual das categorias o requisito será classificado.

4.4 Ficheiros Auxiliares e Dependências

Os métodos desenvolvidos usam ferramentas já implementadas o que origina alguma dependência de certos ficheiros e bibliotecas, de seguida são apresentados os ficheiros e bibliotecas que os métodos implementados precisam para o seu correto funcionamento.

4.4.1 Ficheiros Auxiliares

No que diz respeito ao classificador baseado em métodos formais e PLN, terá como ficheiros auxiliares os modelos de etiquetagem usados *Stanford POS Tagger*, assim como todas as classes que contêm as funcionalidades do mesmo.

O classificador baseado em métodos estatísticos terá como ficheiros auxiliares, os ficheiros que contêm as listas com as palavras mais relevantes. Os ficheiros terão de conter as palavras mais relevantes já etiquetadas e cada linha do texto poderá conter apenas uma única palavra, nunca repetindo as palavras, de modo a facilitar o seu uso pelo classificador.

Na Tabela 4.1 está exemplificado o conteúdo desses ficheiros, na forma de *palavra/etiqueta* para cada uma das listas desenvolvidas, *security* e *dependability*.

Comummente aos dois métodos de classificação, e por opção de implementação, os dois classificadores recebem o texto dos requisitos a implementar por ficheiro, assim sendo, terá de haver sempre um ficheiro de entrada com o texto dos requisitos que se quer classificar.

Palavras mais relevantes: *security*	**Palavras mais relevantes: *dependability***
ensure/VB	fail/VB
authentication/NN	failure/NN
authenticate/NN	average/JJ
certify/VB	probability/NN
secure/JJ	between/IN
security/NN	exceed/VBP
access/NN	surpass/VB
assure/VBP	transcend/JJ
guarantee/NN	maximum/NN
...	...

Tabela 4.1: Exemplo das primeiras linhas das listas com as palavras mais relevantes

4.4.2 Dependências

Os classificadores implementados necessitam de bibliotecas externas para o seu bom funcionamento. Essas bibliotecas estão no formato *Dynamic-link library* (*.dll*) e é imperativo que estejam na mesma diretoria que a biblioteca desenvolvida na altura da execução.

As bibliotecas necessárias à correta execução de ambos os métodos estão definidas na Tabela 4.2.

Biblioteca	**Descrição**	**Usado por**
IKVM	Bibliotecas que possibilitam o uso das funções JAVA do *Stanford POS Tagger*	Ambos os classificadores
stanford-postagger	Biblioteca que contêm as funcionalidades do *Stanford POS Tagger*	Ambos os classificadores
Irony	Bibliotecas referentes ao uso da *Irony - .NET Language Implementation Kit*	Classificador Métodos Formais e PLN

Tabela 4.2: Tabela com a listagem das bibliotecas necessárias para o correto funcionamento dos métodos de classificação

4.5 Conclusões

Neste capítulo foi apresentada a arquitetura desenvolvida no decorrer desta dissertação, foi abordada a estrutura geral, bem como a estrutura dos diferentes componentes que a compõem. Foram apresentados três cenários de utilização desta arquitetura que pensamos poderem beneficiar com um sistema desta natureza e fornecidos vários exemplos de utilização que justificam esta escolha.

Foram apresentados os componentes e as decisões efetuadas, como a escolha das tecnologias utilizadas e as suas justificações.
Foi apresentado o modelo de políticas XML especificadas e a sua integração e aplicação dentro da arquitetura, bem como cada uma das regras implementadas.

O objetivo deste capítulo prende-se à apresentação do funcionamento da arquitetura e à justificação das soluções implementadas nas várias fases que constituem o funcionamento global da mesma.

Capítulo 5

Resultados

5.1 Introdução

Neste capítulo serão apresentados os resultados dos métodos implementado. Serão apresentados os resultados obtidos com os diferentes classificadores, criados no âmbito de uma dissertação desenvolvida consoante o Projeto PROVA, que utiliza a arquitetura proposta.

5.2 Avaliação de Desempenho

De seguida serão apresentados os resultados obtidos com ambos os classificadores, tendo sempre em conta que ambos os classificadores podem sofrer de *overfitting*, isto é, tendo em conta o número reduzido de exemplos disponíveis na criação da gramática que define estruturas de tipos de requisitos funcionais e na recolha de palavras relevantes relativos a requisitos não funcionais de *security* e de *dependability*, poderá ter sido criado um modelo de classificação demasiado adaptado aos exemplos recolhidos.

5.2.1 Classificador Baseado em Métodos Formais e PLN

O classificador baseado em métodos formais e PLN baseia a sua classificação numa gramática que define a estrutura que os requisitos funcionais possuem, do género:

"The system" <verbo_modal> <ação> <descrição_ação>

onde <verbo_modal> normalmente traduz-se por *must*, *should*, ou *shall*; a <ação> define o que é que o sistema deverá realizar, por exemplo, permitir, negar, bloquear, etc; e a <descrição_ação> define o que a <ação> implica, por exemplo, acesso a uma parte do sistema por determinado tipo de utilizador; e foi construída com base num conjunto de treino reduzido que serviu ao mesmo tempo de conjunto de teste.

A Figura 5.1 mostra o final do processo de classificação de um conjunto de requisitos, usando o classificador baseado em métodos formais e PLN onde é atribuída uma categoria ao requisito conforme ele se encaixe na estrutura definida na gramática ou não.

O classificador classifica corretamente todos os requisitos fornecidos, tendo em conta a estrutura dos mesmos. É estimado que o classificador consiga identificar requisitos funcionais dentro da mesma estrutura gramatical que os apresentados na Figura 5.1.

5.2.2 Classificador Baseado em Métodos Estatísticos

O classificador baseado em métodos estatísticos, diferentemente do classificador baseado em métodos formais e PLN, baseia a sua classificação em duas listas de palavras relevantes refe-

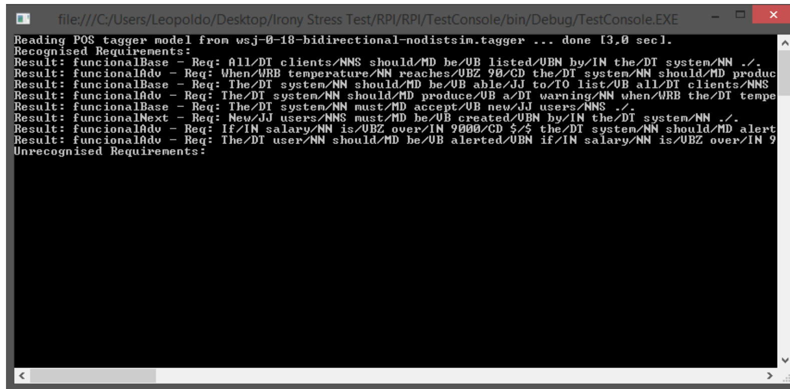

Figura 5.1: Resultado final do Classificador Baseado em Métodos Formais e PLN

rentes a requisitos não funcionais do tipo *security* e *dependability*.

Este classificador compara a frequência das palavras presentes nas listas dos dois tipos de requisitos, e retorna a classificação conforme a maior soma obtida. Apesar de neste momento receber apenas duas listas de palavras relevantes, pode receber todos os tipos de requisitos não funcionais referidos no Capítulo 2, Secção 2.3.2.

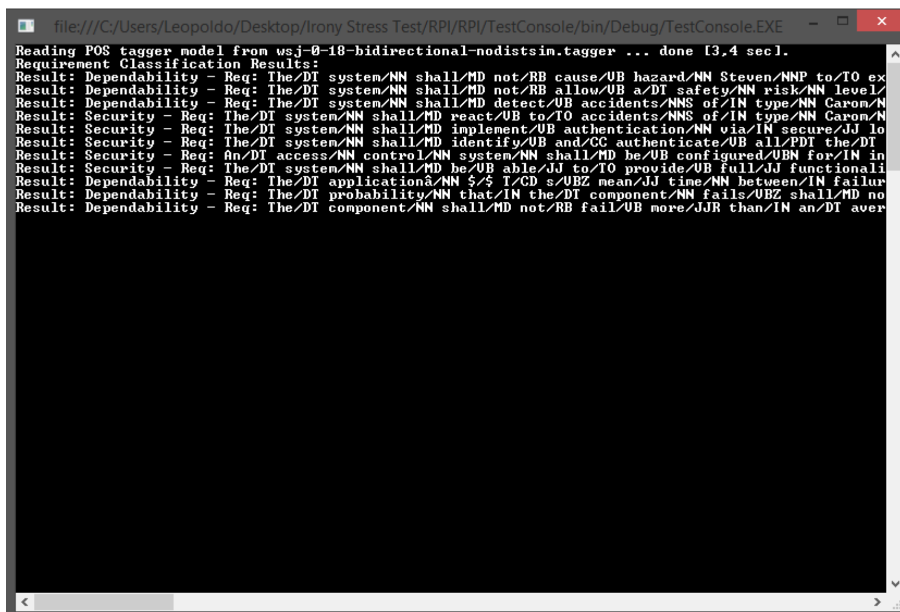

Figura 5.2: Resultado final do Classificador Baseado em Métodos Estatísticos

Este tipo de classificação pode-se tornar mais vantajosa em relação aos métodos formais devido ao facto de não estar dependente de uma estrutura gramatical previamente definida, mas sim de uma lista de palavras facilmente atualizada. O que permite uma maior escalabilidade e precisão da classificação obtida.

Como pode ser observado pela Figura 5.2 o classificador classifica com eficácia todos os requisitos fornecidos, e é estimado que tenha uma eficácia semelhante para requisitos fora do conjunto de testes usado, e com mais listas de palavras relevantes de outras categorias disponibilizadas.

5.3 Conclusões

Neste capítulo foram apresentados os resultados obtidos, nomeadamente os métodos desenvolvidos inseridos numa biblioteca *DLL* e que corresponde à arquitetura proposta no Capítulo 4 e implementação especificada no Capítulo 3.

Em termos de escalabilidade dos métodos de classificação, o classificador estatístico fornece vantagens em relação ao classificador baseado em métodos formais uma vez que é muito mais fácil de recolher palavras relevantes e construir listas com essas mesmas palavras do que a construção de raiz de uma estrutura gramatical para cada tipo de requisito que se quer classificar, requerendo muitos requisitos como exemplo das diferentes categorias para um desenvolvimento devidamente fundamentado.

Comparando a precisão dos dois classificadores, tendo em conta a base em estruturas gramaticais caso o desenvolvimento da estrutura gramatical seja pouco preciso ou com poucos exemplos a estrutura gramatical dos requisitos em geral é muito semelhante. Assim sendo a criação de listas de palavras relevantes pode destacar-se uma vez que é mais precisa e eficaz.

Os tempos de execução obtidos nesta aplicação foram bastante satisfatórios, sendo o *POS Tagger* responsável pela maior parte do tempo decorrido uma vez que é necessário usar um biblioteca convertida da linguagem *JAVA* para *C#*. A etiquetagem de cerca de cento e cinquenta (150) palavras demora, em média 3.3 segundos, sendo que a classificação é um processo rápido, um ou dois segundos no máximo para doze (12) frases propostas a classificação. O que dá um total processamento médio entre quatro (4) a cinco (5) segundos a cada classificação.

Capítulo 6

Conclusão e Trabalho Futuro

6.1 Conclusão

O tema desta dissertação surgiu da parceria entre o laboratório *RELEASE* da Universidade da Beira Interior e a empresa *EDUCED*, no âmbito do projeto PROVA, cujo o objetivo é o desenvolvimento de métodos de classificação automática de texto, em particular a classificação de texto de requisitos de software crítico expresso em linguagem natural. No inicio do trabalho foram identificados e propostos os objetivos a atingir na realização desta dissertação. A investigação e o trabalho levado a cabo durante o decorrer da dissertação permitiu que estes objetivos fossem alcançados com sucesso.

6.1.1 Conclusões

No primeiro capítulo, foi feita uma introdução à temática da dissertação, apresentadas as motivações, indicados os objetivos principais propostos para este trabalho, forma de abordagem à investigação e apresentada a estrutura desta dissertação.

No segundo capítulo, fez-se um levantamento do estado da arte e das tecnologias utilizadas no decorrer do trabalho. Foram apresentados vários conceitos referentes à Engenharia de Requisitos, atividades em que ela se baseia e em qe consiste cada uma delas. Introduziu-se a temática de classificação dos requisitos de software assim como os principais desafios e algumas ferramentas de gestão de requisitos.

No capítulo três foram apresentadas as descrições técnicas dos esquemas implementados e as implementações, que levaram à criação dos métodos de classificação em linguagem *C#* e ao desenvolvimento de uma biblioteca onde eles ficarão disponíveis.

No capítulo seguinte foi apresentada a arquitetura dos métodos de classificação propostos, abordando a estrutura geral dos métodos e os diferentes processos que os compõem, bem como as respetivas justificações para as decisões efetuadas.

Por fim, no último capítulo foram expostos os resultados obtidos,
Os objetivos a alcançar, definidos no início da dissertação, foram cumpridos com sucesso e resultaram nas seguintes contribuições:

- Estudo sobre o processo de recolha e análise de requisitos de software e soluções existentes;
- Estudo sobre os métodos de interpretação de texto baseado em métodos formais e estatísticos;
- Definição de uma gramática onde foi identificada a estrutura de requisitos funcionais, de modo a serem identificados automaticamente;

- Definição de listas de relevância de palavras para requisitos não funcionais, mais propriamente *security* e *dependability* de modo a serem identificados automaticamente;
- Implementação de dois classificadores de modo a automatizar a classificação de requisitos de software crítico;
- Criação de uma biblioteca em linguagem *C#*, em que a ideia é fornecer uma espécie de plugin que possa ser utilizado por outras aplicações;

Na base da arquitetura criada, está um longo trabalho de investigação, que permitiu criar métodos de classificação que permitem automatizar o processo de classificação que ainda é um processo manual e que precisa de algum tempo e dedicação.

Concluindo, os resultados obtidos foram bastante satisfatórios e confiamos que a implementação prática dos métodos desenvolvidos em contexto real, pode ser vantajosa, em relação aos sistemas de gestão e armazenamento de requisitos tradicionais.

A nível pessoal, o trabalho desenvolvido durante o decorrer da dissertação foi bastante gratificante, uma vez que permitiu o contacto com uma variedade de conceitos e tecnologias que eram numa fase inicial desconhecidos e possibilitou uma aprendizagem importante nas áreas de Especificação de Requisitos, Métodos Formais e Processamento de Linguagem Natural.

6.2 Trabalho Futuro

Em termos de trabalho futuro e futuras direções de investigação podem-se dividir em dois aspetos: novas funcionalidades e aspetos de implementação.

6.2.1 Aspetos de Implementação

No que diz respeito aos aspetos de implementação propõe-se:

- Dinamização do processo de carregamento da gramática para o método baseado em métodos formais e PLN, uma vez que neste momento uma nova gramática terá de ser compilada dentro da nova biblioteca;
- Dinamização da recolha de novas palavras relevantes, uma vez que o método de recolha necessita de muitos exemplos para ser eficaz;
- Especificar novas gramáticas para novos tipos de requisitos, para complementação de uma gramática geral, ou criação de várias individuais;
- Possível interação entre os dois métodos de classificação, uma troca de informação ou até comparação de classificações antes de uma decisão final;
- Permitir a classificação de mais tipos de requisitos por parte do classificador baseado em métodos formais, uma vez que para efeitos de teste o classificador apenas classifica requisitos funcionais;
- Permitir a classificação de mais tipos de requisitos por parte do classificador baseado em métodos estatísticos, visto que para efeitos de, classifica somente requisitos do tipo *security* e *dependability*.

6.2.2 Novas Implementações

Como novas funcionalidades propõe-se:

- Tratamento de erros, quando um requisito não consegue ser classificado pelo classificador baseado métodos formais e PLN, é retornado um erro onde a gramática falhou na identificação, esse erro poderá a ser melhorado, de modo a o erro ser mais percetível;
- Numa implementação para classificação enquanto se escreve o texto do requisito, poderá ser sugerida a palavra seguinte, consoante a estrutura gramatical definida;
- Método de recolha de informações sobre os tipos de requisitos no funcionamento do software, de modo a recolher informações para melhoramento da gramática e palavras mais relevantes;
- Possibilidade de implementação de uma aplicação de classificação independente, onde faria somente classificação de requisitos.

Bibliografia

[AU10] Dr. Sohail Asghar and Mahrukh Umar. Requirement engineering challenges in development of software applications and selection of customer-off-the-shelf (cots) components. http://www.cscjournals.org/csc/manuscript/Journals/IJSE/volume1/Issue2/IJSE-7.pdf, July 2010. 12

[BB10] Laurel J. Brinton and Donna M. Brinton, 2010. 20

[Boe91] B.W. Boehm. Software risk management: principles and practices, Jan 1991. 10

[BR00] Keith H. Bennett and Václav T. Rajlich. Software maintenance and evolution: A roadmap, 2000. Available from: http://doi.acm.org/10.1145/336512.336534. 10

[BS09] Mohammad U. Bokhari and Shams T. Siddiqui. A comparative study of software requirements tools for secure software development, February 2009. 13

[Cai14] Eric Cai. Machine learning lesson of the day - overfitting and underfitting. http://www.statsblogs.com/2014/03/20/machine-learning-lesson-of-the-day-overfitting-and-underfitting/, 2014. 20

[Cha87] Nigel P. Chapman. Lr parsing: Theory and practice, 1987. 21

[Chu93] Lawrence Chung. Dealing with security requirements during the development of information systems, 1993. Available from: http://dl.acm.org/citation.cfm?id=646081.676526. 8

[Dav92] Alan M. Davis. Operational prototyping: A new development approach, September 1992. Available from: http://dx.doi.org/10.1109/52.156899. 7

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition, April 1993. Available from: http://dx.doi.org/10.1016/0167-6423(93)90021-G. 6, 8

[DWT12] Alan Dennis, Barbara Haley Wixom, and David Tegarden. Systems analysis and design with uml, 2012. 11

[FN88] Stephen Fickas and P. Nagarajan. Critiquing software specifications, November 1988. Available from: http://dx.doi.org/10.1109/52.10002. 9

[Fri11] Jeroen Frijters. Ikvm homepage. http://www.ikvm.net/#Introduction, 2011. 17, 18

[GF94] O. C Z Gotel and A C W Finkelstein. An analysis of the requirements traceability problem, Apr 1994. 9

[GGP01] B Gomolski, J Grigg, and K. Potter. 2001 it spending and staffing survey results. http://www.isixsigma.com/industries/software-it/lowering-cost-ownership-software-lifecycles/, 2001. 1

[GH96] A Gravell and P. Henderson. Executing formal specifications need not be harmful, Mar 1996. 9

[GL93] Joseph A Goguen and C. Linde. Techniques for requirements elicitation, Jan 1993. 7

[GN99] Carlo Ghezzi and Bashar Nuesibeh. Guest editorial: Introduction to the special section, November 1999. Available from: http://dx.doi.org/10.1109/TSE.1999.824393. 10

[GNCC98] Diego Del Gobbo, Marcello Napolitano, John Callahan, and Bojan Cukic. Experience in developing system requirements specification for a sensor failure detection and identification scheme, 1998. Available from: http://dl.acm.org/citation.cfm?id=645432.652405. 8

[Gog94] Joseph A. Goguen. Requirements engineering, 1994. Available from: http://dl.acm.org/citation.cfm?id=177970.184582. 6

[Gou14] John Gough. Gardens point lex. https://gplex.codeplex.com/, 2014. 21

[HN95] Anthony Hunter and Bashar Nuseibeh. Managing inconsistent specifications: Reasoning, analysis, and action, 1995. 10

[Hol97] Gerard J. Holzmann. The model checker spin, May 1997. Available from: http://dx.doi.org/10.1109/32.588521. 9

[Iva14] Roman Ivantsov. Irony - .net language implementation kit. https://irony.codeplex.com/, 2014. 22

[Joh92] Peter Johnson. Human-computer interaction : psychology, task analysis and software engineering, 1992. 8

[JZ93] M. Jackson and P. Zave. Domain descriptions, Jan 1993. 8

[Kov03] B. Kovitz. Why classify requirements. http://c2.com/cgi/wiki?WhyClassifyRequirements, 2003. 2

[Leh80] M.M. Lehman. Programs, life cycles, and laws of software evolution, Sept 1980. 9

[LHM+98] Robyn R. Lutz, Guy G. Helmer, Michelle M. Moseman, David E. Statezni, and Stephen R. Tockey. Safety analysis of requirements for a product family, 1998. Available from: http://dl.acm.org/citation.cfm?id=645536.657172. 8

[LK95] P. Loucopoulos and E. Kavakli. Enterprise modelling and the teleological approach to requirements engineering, 1995. 8

[LMB95] John R. Levine, Tony Mason, and Doug Brown, 1995. 21

[MLR+97] F. Modugno, N.G. Leveson, J.D. Reese, K. Partridge, and S.D. Sandys. Integrated safety analysis of requirements specifications, Jan 1997. 8

[MR96] N.A.M. Maiden and G. Rugg. Acre: selecting methods for requirements acquisition, May 1996. Available from: http://digital-library.theiet.org/content/journals/10.1049/sej.1996.0024. 7

[MTM90] Mitchell Marcus, Ann Taylor, and Robert MacIntyre. The penn treebank project. http://www.cis.upenn.edu/t̃reebank/, 1990. 16

[Net14] Microsoft Developer Network. Exception class documentation. http://msdn.microsoft.com/en-us/library/system.exception$v = vs.110$.aspx, 2014. 28

[Nor] Richard Nordquist. lexical-functional grammar. http://grammar.about.com/od/il/g/Lexical-Functional-Grammar-lfg.htm. 18

[Nor14] Richard Nordquist. about education: parsing. http://grammar.about.com/od/pq/g/parsingterm.htm, 2014. 15

[Saa97] Mark Saaltink. The z/eves system, 1997. Available from: http://dl.acm.org/citation.cfm?id=647282.722913. 8

[SFG99] Helen Sharp, Anthony Finkelsteiin, and Galal Galal. Stakeholder identification in the requirements engineering process, 1999. Available from: http://dl.acm.org/citation.cfm?id=519627.790338. 6

[SG96] M.L.G. Shaw and B.R. Gaines. Requirements acquisition, May 1996. 7

[Shi09] Chigusa Shinichi. Parsing: in depth. http://www.sal.tohoku.ac.jp/ling/corpus2/2full.htm, 2009. 16

[Som10] Ian Sommerville. Software engineering (9th edition), 2010. 10, 11

[Tih13a] Sergey Tihon. Nlp: Stanford pos tagger with f# $.net$. http://sergeytihon.wordpress.com/2013/02/08/nlp-stanford-pos-tagger-with-f-net/, 2013. 17

[Tih13b] Sergey Tihon. Sergey tihon - about me. http://about.me/sergey.tihon, 2013. 17

[Tih14] Sergey Tihon. Stanford log-linear part-of-speech tagger for .net. http://sergey-tihon.github.io/Stanford.NLP.NET/StanfordPOSTagger.html, 2014. 17

[TKMS14] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Stanford log-linear part-of-speech tagger homepage. http://nlp.stanford.edu/software/tagger.shtml, 2014. 16

[Tur08] N. Turbit. Key stakeholder support. http://www.projectperfect.com.au/info_key_stakeholder.php, 2008. 13

[VS99] Stephen Viller and Ian Sommerville. Social analysis in the requirements engineering process: From ethnography to method, 1999. Available from: http://dl.acm.org/citation.cfm?id=647646.731255. 7

[Zav97] Pamela Zave. Classification of research efforts in requirements engineering, December 1997. Available from: http://doi.acm.org/10.1145/267580.267581. 5

Apêndice A

Anexos

A.1 Anexo A

```
public GramaticaFuncional() : base(false)
{
  var funcionalList = new NonTerminal("funcionalList");

  var funcional = new NonTerminal("funcional");
  var palavra = new IdentifierTerminal("palavra");

  var pos = new NonTerminal("pos");
  pos.Rule = ToTerm("CC") | ToTerm("CD") | ToTerm("DT") | ToTerm("EX") | ToTerm("FW") | ToTerm("IN"↩
      ) | ToTerm("JJ") | ToTerm("JJR") | ToTerm("JJS") | ToTerm("LS") | ToTerm("MD") | ToTerm("NN"↩
      ) | ToTerm("NNS") | ToTerm("NNP") | ToTerm("NNPS") | ToTerm("PDT") | ToTerm("POS") | ToTerm(↩
      "PRP") | ToTerm("PRP$") | ToTerm("RB") | ToTerm("RBR") | ToTerm("RBS") | ToTerm("RP") | ↩
      ToTerm("SYM") | ToTerm("TO") | ToTerm("UH") | ToTerm("VB") | ToTerm("VBD") | ToTerm("VBG") |↩
       ToTerm("VBN") | ToTerm("VBP") | ToTerm("VBZ") | ToTerm("WDT") | ToTerm("WP") | ToTerm("WP$"↩
      ) | ToTerm("WRB") | ToTerm("$");

  var pontuacao = new NonTerminal("pontuacao");
  pontuacao.Rule = ToTerm(".") | ToTerm("?") | ToTerm("!") | ToTerm(":") | ToTerm(";") | ToTerm("—"↩
      ) | ToTerm("_") | ToTerm("(") | ToTerm(")") | ToTerm("[") | ToTerm("]") | ToTerm("...") | ↩
      ToTerm("'") | ToTerm("\"") | ToTerm(",");

  var simbolo = new NonTerminal("simbolo");
  simbolo.Rule = palavra + "/$" |
  ToTerm("$/$")
  ;

  var numero = new NumberLiteral("numero");
  var valor = new NonTerminal("valor");
  valor.Rule = numero + "/CD"
  ;

  var valorSimb = new NonTerminal("valorSimb");
  valorSimb.Rule = valor + simbolo |
  simbolo + valor
  ;

  var unidadePontuac = new NonTerminal("upontuac");
  unidadePontuac.Rule = pontuacao + "/" + pontuacao;


  var unidade = new NonTerminal("unidade");
  unidade.Rule = unidadePontuac |
  palavra + "/" + pos |
  valorSimb;

  var frase = new NonTerminal("frase");
  frase.Rule = unidade |
  frase + unidade;

  var determinante = new NonTerminal("determinante");
  var nomes = new NonTerminal("nomes");
  var nome = new NonTerminal("nome");
  var modal = new NonTerminal("modal");
  var accao = new NonTerminal("accao");
  var adverbio = new NonTerminal("adverbio");
```

```
var verbo = new NonTerminal("verbo");
var verboM = new NonTerminal("verboM");
var verboAux = new NonTerminal("verboAux");
var preposicao = new NonTerminal("preposicao");
var condicionante = new NonTerminal("condicionante");
var adjetivo = new NonTerminal("adjetivo");
var adjetivoTo = new NonTerminal("adjetivoTo");
var complementoDireto = new NonTerminal("complementoDireto");
var posNN = new NonTerminal("posNN");
var posVB = new NonTerminal("posVB");
var posRB = new NonTerminal("posRB");
var posJJ = new NonTerminal("posJJ");
var posTO = new NonTerminal("posTO");


var funcionalBase = new NonTerminal("funcionalBase");
var funcionalAdv = new NonTerminal("funcionalAdv");
var funcionalNext = new NonTerminal("funcionalNext");

this.Root = funcional;

funcional.Rule = funcionalBase | funcionalAdv | funcionalNext;

funcionalBase.Rule = determinante + nomes + modal + accao + (unidadePontuac | Empty )
;

funcionalAdv.Rule = funcionalBase + funcionalNext |
adverbio + nomes + verbo + valorSimb + funcionalBase |
adverbio + nomes + verbo + valor + funcionalBase |
preposicao + nomes + verbo + condicionante + funcionalBase |
adjetivo + nomes + modal + verbo + condicionante + funcionalBase
;

funcionalNext.Rule = adverbio + determinante + nomes + verbo + valorSimb + unidadePontuac |
adverbio + determinante + nomes + verbo + valor + unidadePontuac |
adverbio + nomes + verbo + valorSimb + unidadePontuac |
adverbio + nomes + verbo + valor + unidadePontuac |
preposicao + nomes + verbo + condicionante + unidadePontuac |
adjetivo + nomes + modal + verbo + condicionante + unidadePontuac |
verbo + condicionante + unidadePontuac
;

posNN.Rule = ToTerm("NN") | ToTerm("NNS") | ToTerm("NNP") | ToTerm("NNPS");
nome.Rule = palavra + "/" + posNN;

nomes.Rule = nome
;

accao.Rule = complementoDireto |
(verbo + adjetivo) + nomes |
verbo + adjetivoTo + nomes |
(verbo + adjetivo) + complementoDireto |
verbo + adjetivoTo + complementoDireto |
(verbo + adjetivo + preposicao) + complementoDireto |
verbo + adjetivoTo + preposicao + complementoDireto
;

posVB.Rule = ToTerm("VB") | ToTerm("VBD") | ToTerm("VBG") | ToTerm("VBN") | ToTerm("VBP") | ↩
    ToTerm("VBZ");
verbo.Rule = palavra + "/" + posVB;
verboAux.Rule = (ToTerm("be") + "/" + posVB);

verboM.Rule = ToTerm("should") | ToTerm("shall") | ToTerm("must");
modal.Rule = verboM + "/MD" |
verboM + "/MD" + verboAux
;

determinante.Rule = palavra + "/DT";

posTO.Rule = "/TO";
```

```
  adjetivo.Rule = palavra + "/" + posJJ
  ;
  adjetivoTo.Rule = adjetivo + palavra + posTO
  posJJ.Rule = ToTerm("JJ") | ToTerm("JJR") | ToTerm("JJS");

  complementoDireto.Rule = verbo + nomes |
  adjetivo + nomes |
  adjetivoTo + nomes |
  adjetivo + verbo + determinante + nomes |
  adjetivoTo + verbo + determinante + nomes |
  determinante + nomes |
  verbo + determinante + nomes |
  verbo + preposicao + determinante + nomes |
  verbo + preposicao + nomes
  ;

  posRB.Rule = ToTerm("RB") | ToTerm("RBR") | ToTerm("RBS") | ToTerm("WRB");
  adverbio.Rule = palavra + "/" + posRB;

  preposicao.Rule = palavra + "/IN";

  condicionante.Rule = preposicao + valorSimb |
  preposicao + determinante + nomes;
}
```

A.2 Anexo B

```
public class Main
{
  public List<string> inputRequirements;
  public List<Result> outputRequirements;

  public Main()
  {
    inputRequirements = new List<string>();
    outputRequirements = new List<Result>();
  }

  public void ProcessRequirements(string filename, Grammar grammar)
  {

    if (grammar == null)
    {
      throw new RPIException("Grammar not defined. Exiting..");
    }

    Parser p = new Parser(grammar);
    ParseTree tree;

    inputRequirements = POSTagger.TagFile(filename);

    if (inputRequirements.Count == 0)
    {
      throw new RPIException("Tagger error, check for anomaly.");
    }

    foreach (string item in inputRequirements)
    {
      tree = p.Parse(item.ToString());

      if (tree.HasErrors())
      {
        foreach (var itemError in tree.ParserMessages)
        {
          outputRequirements.Add(new Result(item, null, itemError.Message));
        }
      }
```

```
        else
        {
          foreach (var result in tree.Root.ChildNodes)
          {
            outputRequirements.Add(new Result(item, result.Term.ToString(), null));
          }
        }
      }
    }
}
```

A.3 Anexo C

```
public string process(string text)
{
  string category = string.Empty;

  freqDep.Clear();
  freqSec.Clear();

  freqXonT(securityMCW, text, freqSec);
  freqXonT(dependabilityMCW, text, freqDep);

  double sumSecurity = sumFreqX(freqSec, wordCount(text));
  double sumDependability = sumFreqX(freqDep, wordCount(text));

  if (sumSecurity == sumDependability)
  {
    category = "undefined";
  }
  else
  {
    if (sumSecurity > sumDependability)
    {
      category = "security";
    }
    if (sumDependability > sumSecurity)
    {
      category = "dependability";
    }
  }

  return category;
}
```

A.4 Anexo D

```
public class Result
{
  public string reqText {  get; set; }
  public string result { get; set; }
  public string errorMessage { get; set; }

  public Result(string text, string classification, string message)
  {
    reqText = text;
    result = classification;
    errorMessage = message;
  }
}
```

Glossário

LaTeX	Conjunto de macros para o processador de textos TeX, utilizado amplamente para a produção de textos matemáticos e científicos devido à sua alta qualidade tipográfica.
TRS	*Technical Requirement Specification* define-se como uma descrição detalhada dos requisitos técnicos, com os critérios de aceitação específicos, expressos em termos adequados para formar uma base para os processos de desenvolvimento do projeto e de produção real para que um item tenha as qualidades especificadas nas características operacionais.
Interface Externa	Componente de um software que permite a outros softwares ou utilizadores a interação com o sistema a que pertence.
Glossário	Em engenharia de requisitos, define-se glossário como uma lista de palavras chave que pertencem a um conjunto requisitos, identificando quais deles contêm essas mesmas palavras.
Prototipagem	Define-se prototipagem como a criação de uma demonstração de como o sistema irá funcionar