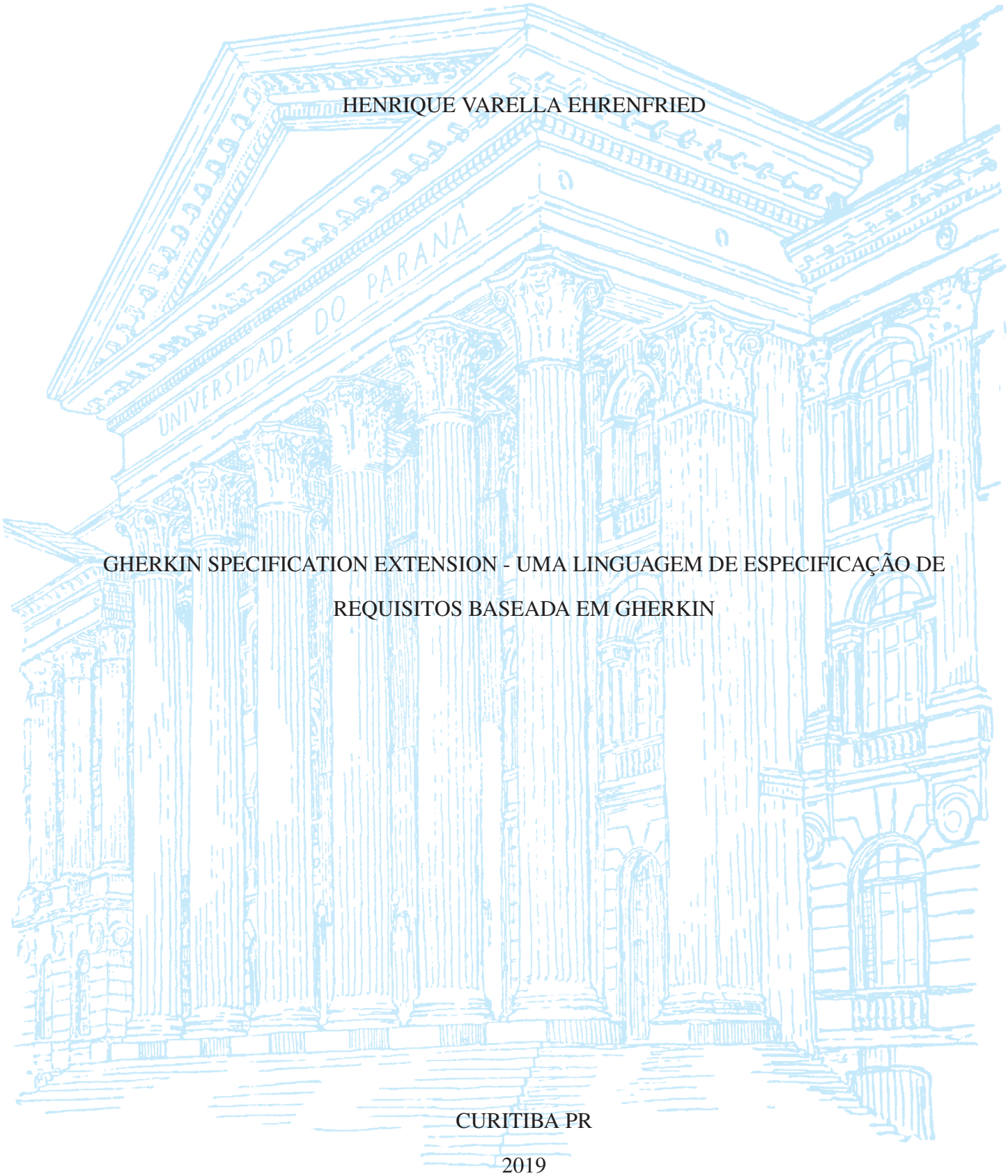
UNIVERSIDADE FEDERAL DO PARANÁ

HENRIQUE VARELLA EHRENFRIED



GHERKIN SPECIFICATION EXTENSION - UMA LINGUAGEM DE ESPECIFICAÇÃO DE REQUISITOS BASEADA EM GHERKIN

CURITIBA PR

2019

HENRIQUE VARELLA EHRENFRIED

GHERKIN SPECIFICATION EXTENSION - UMA LINGUAGEM DE ESPECIFICAÇÃO DE REQUISITOS BASEADA EM GHERKIN

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Andrey Ricardo Pimentel.

CURITIBA PR

2019

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

E33g

Ehrenfried, Henrique Varella
Gherkin Specification Extension - Uma linguagem de especificação de requisitos baseada em Gherkin [recurso eletrônico] / Henrique Varella Ehrenfried. – Curitiba, 2019.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2019.

Orientador: Andrey Ricardo Pimentel .

1. Processamento de linguagem natural (Computação). 2. Software – Desenvolvimento. 3. Inteligência artificial. I. Universidade Federal do Paraná. II. Pimentel, Andrey Ricardo. III. Título.

CDD: 001.535

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

MINISTÉRIO DA EDUCAÇÃO
SETOR SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA - 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **HENRIQUE VARELLA EHRENFRIED** intitulada: **Gherkin Specification Extension - Uma linguagem de especificação de requisitos baseada em Gherkin**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua aprovação no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 28 de Fevereiro de 2019.

ANDREY RICARDO PIMENTEL
Presidente da Banca Examinadora (UFPR)

ROBERTO PEREIRA
Avaliador Interno (UFPR)

ADOLFO GUSTAVO SERRA SECA NETO
Avaliador Externo (UTFPR)

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Aos meus pais que sempre estiveram ao meu lado

AGRADECIMENTOS

Agradeço a meus pais, que sempre me orientaram e apoiaram. Agradeço também meu orientador, professor Andrey R. Pimentel que me auxiliou durante o desenvolvimento deste trabalho. Agradeço ao professor Roberto Pereira que, em minha qualificação, forneceu sugestões sobre métodos de avaliação. Agradeço à professora Natasha M. C. Valentim, que me ajudou a entender a técnica utilizada no experimento deste trabalho, assim como no processo de experimentação de campo. Agradeço a todos que se dispuseram de seu tempo para participar de meus estudos de casos, e todos que, direta ou indiretamente, participaram desta pesquisa.

RESUMO

O desenvolvimento de software é composto de várias fases, entre elas está a fase de elicitação, negociação e validação de requisitos. Nesta fase, geralmente utiliza-se linguagem natural para definir e negociar os requisitos do sistema que será desenvolvido. Entretanto, as linguagens naturais podem ser ambíguas, dificultando o entendimento do requisito, e portanto a sua negociação e validação. Uma das tentativas de solucionar o problema da ambiguidade foi a criação de linguagens de especificação de requisitos. Algumas destas linguagens, como Z e Alneelain, utilizam métodos formais na definição dos requisitos. O problema da utilização de métodos formais é que todos que lerão os requisitos devem ter conhecimentos em métodos formais, fato que nem sempre é verdade, já que alguns interessados pelo software, como clientes e analistas de negócios, podem não ter este conhecimento. Outras linguagens, como i* e KAOS, utilizam elementos gráficos para especificar um sistema. Este formato pode complicar o entendimento, pois diagramas grandes tem alta complexidade cognitiva. Para tentar solucionar este problema, é apresentado o Gherkin Specification Extension (GSE), uma extensão da linguagem Gherkin, que utiliza linguagem natural em conjunto de estruturas fixas para reduzir o problema da ambiguidade durante a especificação de requisitos utilizando linguagem natural. O GSE foi estruturado com base em onze requisitos que definem sete seções diferentes (Descrição, Grupo, Restrições e qualidades, Relacionamento, Planejamento, Métricas e Notas). Este requisitos foram estudados de forma a identificar funcionalidades chave para melhorar a comunicação entre diversos interessados no software, seja ele um desenvolvedor, gerente de projetos, cliente, usuário ou analista de negócios. A extensão foi validada quanto a sua aceitação com pessoas com poucos conhecimentos em desenvolvimento de software, obtendo resultado positivos referente à qualidade da especificação gerada e aceitação da tecnologia. Para mensurar a aceitação da tecnologia foi utilizado o modelo TAM (Technology Acceptance Model) em sua terceira versão.

Palavras-chave: Linguagem, Especificação, Requisitos

ABSTRACT

Software development consists of several phases, including elicitation, negotiation and validation of requirements. At this stage, natural language is usually used to define and negotiate the system requirements that will be developed. However, natural languages can be ambiguous, making it difficult to understand the requirement, and therefore its negotiation and validation. One of the attempts to solve the ambiguity problem was the creation of requirements specification languages. Some of these languages, such as Z and Alneelain, use formal methods in defining requirements. The problem with using formal methods is that everyone who will read the requirements must have knowledge of formal methods, a fact that is not always true, as some software stakeholders, such as customers and business analysts, may not have this knowledge. Other languages, such as i * and KAOS, use graphics to specify a system. This format may complicate understanding, since very large diagrams generally have high cognitive complexity. In order to solve this problem, the Gherkin Specification Extension (GSE), an extension of the Gherkin language, is presented, which uses natural language in conjunction with fixed structures to reduce the problem of ambiguity during specification of requirements using natural language . The GSE was structured based on eleven requirements that define seven different sections (Description, Group, Constraints and Qualities, Relationship, Planning, Metrics and Notes). These requirements have been studied in order to identify key functionalities to improve communication among diverse stakeholders in the software, be it a developer, project manager, client, user or business analyst. The extension was validated for its acceptance with people with little knowledge in software development, obtaining positive results regarding the quality of the generated specification and acceptance of the technology. To measure the acceptance of the technology, the TAM (Technology Acceptance Model) model was used in its third version.

Keywords: Language, Specification, Requirement

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
INVEST	Independable, Negotiable, Valuable, Estimable, Small, Testable
BDD	Behavior Driven Development
TDD	Test Driven Development
VDM	Vienna Development Method
UML	Unified Model Language
CSV	Comma Separated Values
JSON	JavaScript Object Notation

SUMÁRIO

1 INTRODUÇÃO

Desenvolver um software que atenda ao propósito para o qual ele foi desenvolvido é essencial para que ele seja um sucesso (Nuseibeh e Easterbrook, 2000). Um dos preocessos que podem ser utilizadas para alcançar este objetivo é a engenharia de requisitos (ur Rehman et al., 2013).

A engenharia de requisitos é o campo do conhecimento que estuda formas de obter e gerenciar os requisitos de um projeto(ur Rehman et al., 2013). Os requisitos são importantes para que haja uma comunicação entre interessados em um projeto, de forma que a equipe de desenvolvimento defina o que os usuários e clientes querem que seja desenvolvido. Usualmente utiliza-se linguagem natural para comunicar os requisitos de um projeto (Widrig e Leffingwell, 1999) (da Silva Xavier, 2014). Esta escolha geralmente é feita porque a linguagem natural já é conhecida pelos clientes e usuários, possibilitando uma participação mais ativa dos clientes e usuários no desenvolvimento de software. Esta participação pode ocorrer de diversas formas como por exemplo no questionamento de requisitos que não ficaram claros durante o desenvolvimento de software, na definição do problema que o software deve resolver e na validação dos requisitos do software.

Neste Capítulo, será exposto o problema que será tratado por este trabalho. A motivação desta pesquisa é descrita na Seção 1.1. Os objetivos desta pesquisa serão expostos na Seção 1.2. Por fim a Seção 1.3 define como este documento está estruturado.

1.1 MOTIVAÇÃO

Como os usuários e clientes tem o papel importante de esclarecer e validar o que deve ser feito durante o desenvolvimento de software, é preciso que haja uma comunicação efetiva entre desenvolvedores, clientes e usuários. E portanto, estes três interessados no desenvolvimento do software devem ser capazes de entender a linguagem que utilizarão.

A linguagem natural, utilizada na definição de requisitos e no esclarecimento de dúvidas dos desenvolvedores, pode introduzir problemas de comunicação, pois é possível criar sentenças ambíguas com linguagem natural, levando ao mau entendimento do requisito proposto.

Uma sentença ambígua é por exemplo a sentença "Thiago pediu a Daniel para assinar o contrato". Nesta sentença, não é possível saber se 1) "Thiago pediu que Daniel assinasse o contrato" ou se 2) "Thiago pediu permissão a Daniel para assinar o contrato". Perceba que, em 1, quem assina o contrato é Daniel, em 2 quem assina o contrato é Thiago.

Para tentar evitar o problema da ambiguidade, linguagens de especificação de requisitos como RSLingo (De et al., 2012), Z (Davies e Woodcock, 1996), Alneelain (Ali et al., 2017) e outras surgiram. Entretanto elas possuem outro problema: a sua utilização pelos clientes e usuários pode ser difícil, já que muitas destas linguagens como Z e Alneelain, apesar de reduzir as ambiguidades, utilizam métodos formais e, portanto, exigem um conhecimento prévio considerável. A necessidade deste conhecimento prévio pode ser um problema, pois pode acontecer de um ou mais dos interessados no desenvolvimento do software não possuir este conhecimento, tornando a comunicação inefetiva, uma vez que nem todos os interessados no desenvolvimento do software podem não estar dispostos a aprender linguagens formais.

1.2 OBJETIVOS

O objetivo desta pesquisa é desenvolver uma forma de especificar software de forma mais precisa e acessível a diversos tipos de usuários, para que a comunicação entre os interessados no desenvolvimento do software tenha menos ruídos. Para tanto será proposto o Gherkin Specification Language (GSE), o qual é uma extensão do Gherkin, uma linguagem utilizada em especificação de testes de aceitação que para tanto utiliza um formato de História de Usuário para definir os cenários que deverão ser testados.

1.2.1 Objetivo geral

O objetivo deste trabalho é desenvolver uma linguagem ou extensão para uma linguagem, com o foco na aceitação da tecnologia por todos os interessados, desta forma permitindo uma comunicação melhor entre os interessados em um projeto de software.

1.2.2 Objetivos específicos

Para atingir o objetivo geral do trabalho será preciso atingir os seguintes objetivos específicos.

- Uma pesquisa sobre o estado da arte deverá ser produzida
- Uma gramática para a extensão, seguindo o padrão do Gherkin deverá ser desenvolvida
- Um interpretador para a extensão deverá ser desenvolvido
  - O interpretador deverá ser capaz de gerar um relatório a partir dos requisitos especificados na linguagem GSE
- A facilidade de uso do GSE por pessoas que não têm experiência em desenvolvimento de software deverá ser testada
  - Uma pesquisa de campo com possíveis utilizadores do GSE deverá ser realizada

1.3 ORGANIZAÇÃO DO DOCUMENTO

Este documento é dividido de forma que os conceitos necessários para o seu entendimento são explicados no Capítulo 2. Os trabalhos relacionados, contendo outras linguagens de especificação de requisitos, são apresentados no Capítulo 3. Os requisitos do GSE são definidos e explicados no Capítulo 4. A arquitetura do GSE e de seu interpretador são mostrados no Capítulo 5. O estudo de caso conduzido para testar a aceitação do GSE pelo usuário que não pertence à área da computação é exposto no Capítulo 6. Encerrando este trabalho, é apresentada a conclusão do trabalho no Capítulo 7.

2 CONCEITOS

Neste Capítulo serão expostos conceitos necessários para a compreensão da proposta do projeto de mestrado. Começando com a Seção 2.1 que elucida os conceitos de *stakeholder*. Em sequência, a Seção 2.2 apresenta o que é um requisito e como eles podem ser classificados. A Seção 2.3 explica sobre Histórias de Usuário. A Seção 2.4 expõe os conceitos básicos sobre o processo de desenvolvimento *Behavior Driven Development*. Na Seção 2.5 é exposta a ferramenta de testes de aceitação *Cucumber*, a qual utiliza a gramática do *Gherkin*, que é detalhada na Seção 3.1. Por fim, a Seção 2.6 apresenta a conclusão deste Capítulo.

2.1 *STAKEHOLDER*

Há várias definições para *stakeholders*. Segundo Freeman (2004), os *stakeholders* são definidos como *"Those groups who are vital to the survival and succes of the organization"* (Tradução livre: Aqueles grupos que são vitais para a sobrevivência e sucesso da organização). Entretanto a definição mais aceita nos círculos acadêmicos é a de Freeman (1984). Para ele stakeholders são *"Any group or individual who can affect or is affected by the achievement of the organization objectives"* (Tradução livre: Qualquer grupo ou indivíduo que pode afetar ou é afetado por uma organização alcançar seus objetivos).

Desta forma, Friedman e Miles (2006) apontam que a parte da sentença que diz "afeta ou são afetados" permite a interpretação de que indivíduos e grupos fora da organização podem ser considerados como stakeholders da organização, mesmo que a organização não considere este indivíduo ou grupo como seu *stakeholder*.

O *stakeholder* alvo deste projeto, também chamado neste trabalho de usuário não técnico, nem sempre terá conhecimentos em desenvolvimento de software. Portanto, é importante que exista uma forma simples de comunicação entre cliente, usuário e desenvolvedor. Note que embora usuários e clientes pareçam a mesma entidade, nem sempre eles são. O dicionário Michaelis[1] define usuário como sendo: "1. Que utiliza algo; que tem o direito de uso, mas não a propriedade. 2. Que serve ou que é próprio para nosso uso". O mesmo dicionário Michaelis[2] define cliente como sendo: "[...]3. Comprador frequente de um estabelecimento comercial; freguês". Segundo B. Nuseibeh (2007), apenas em casos especiais um cliente é um usuário.

A forma que usualmente é utilizada na comunicação entre stakeholders é a escrita de textos em linguagem natural. Porém a linguagem natural tem uma grande desvantagem, ela pode ser ambígua, como por exemplo na frase "Beatriz encontrou o gerente da loja com o seu amigo", em que pode ser interpretado que a Beatriz e seu amigo encontram o gerente da loja **ou** que Beatriz encontrou o gerente da loja com o amigo dele. Portanto, o uso da linguagem natural poderá, apenas considerando ambiguidade, introduzir maus entendimentos na especificação do software. Se duas interpretações distintas do software definido forem obtidas no cenário descrito pela Figura 2.1, há uma grande possibilidade de que o sistema desenvolvido não atenda as expectativas do cliente tampouco as do usuário.

Suponha que no cenário onde o cliente não é o usuário, assim como mostra a Figura 2.1, existe o cliente (CL), o usuário (US) e a equipe de desenvolvedores (DV). O CL é o cliente dos DV

[1] `https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/Usu%C3%A1rio/` - Acessado em 13/12/2018.

[2] `https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/Cliente/` - Acessado em 13/12/2018.

e US são os usuários do sistema que CL quer encomendar de DV. Para saber os requisitos de US, CL se comunica com US e prospecta o que o sistema deverá ser capaz de fazer. Então, em uma reunião para contratar os serviços de DV, CL precisa explicar todas as funcionalidades que US requisitou. Neste cenário, US utilizou linguagem natural para se comunicar com CL, entretanto CL também utiliza linguagem natural para explicar o sistema a ser desenvolvido por DV. Esta troca de informações entre US e CL, depois de CL e DV podem gerar mal entendimentos do que o software realmente deve fazer. Para tentar mitigar estes mal entendidos é preciso realizar a validação e negociação dos requisitos entre os desenvolvedores, clientes e usuários (Abran et al., 2004).

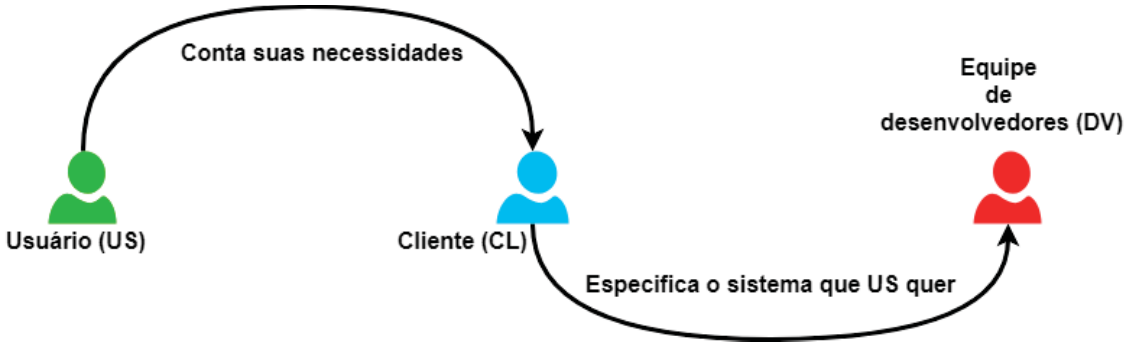


Figura 2.1: Exemplo de fluxo de comunicação entre stakeholders.

Para resolver parcialmente este problema, modelos de História de Usuário surgiram, permitindo que a linguagem natural continue sendo utilizada, mas agora com restrições em um formato pré-definido que contempla as principais informações sobre o software (McDonald, 2016).

No escopo deste projeto, os usuários não técnicos interagem com os desenvolvedores de software. Portanto, a gestão de projetos que faz mais sentido é a dos métodos ágeis. Os métodos ágeis permitem que os projetos sejam adaptados ao longo de seu ciclo de vida, pois estes métodos utilizam entregas incrementais e ciclos iterativos de desenvolvimento com entregas de software mais frequentes, uma vez que geralmente há uma entrega entre 1 e 3 semanas[3]. Os métodos ágeis surgiram com a publicação do manifesto ágil em 2001[4]. Neste manifesto definiu-se que:

- Os Indivíduos e as interações entre eles são mais importantes que processos e ferramentas;
- Um software em funcionamento é mais importante que uma documentação abrangente;
- A colaboração com os clientes é mais importante que a negociação de contratos;
- Responder a mudanças é mais importante que seguir um plano

Note que ser mais importante significa que um valor tem mais prioridade que outros. Portanto mesmo que um artefato não seja considerado "mais importante", ele ainda deve ser feito. Desta forma os métodos ágeis valorizam mais a interação entre as pessoas que seguir todos os detalhes de um método de desenvolvimento.

Como a comunicação entre os stakeholders durante o desenvolvimento do software é essencial para um produto final de qualidade, é preciso um meio para que os stakeholders

[3]Disponível em `https://www.culturaagil.com.br/o-que-sao-metodos-ageis/` Acessado em: 16/03/2018

[4]`http://agilemanifesto.org` Acessado em 19/03/2018

consigam comunicar seus requisitos. É este problema que a criação da extensão de uma linguagem de requisitos tenta resolver, já que um dos princípios do manifesto ágil (K. Beck, 2001) é *"Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto."*.

2.2 REQUISITOS

Requisitos podem ter diversas definições. Entretanto elas geralmente são próximas em significado. Por exemplo, para o dicionário Michaelis[5], um requisito é definido como *"1. Condição ou exigência imprescindível a que se deve satisfazer para alcançar determinado fim. [...]"*. Para Sommerville (2011), requisitos são "O que o software deve fazer", enquanto que B. Nuseibeh (2007) diz que requisitos são "As coisas no mundo que nós gostaríamos de alcançar". Abran et al. (2004) define um requisito de software como "uma propriedade que o software deve exibir para que um problema do mundo real seja resolvido". Widrig e Leffingwell (1999) definem um requisito de duas formas: "A capacidade necessária do software para que o usuário resolva um problema ou alcance um objetivo"e "Uma que o software deve possuir ou realizar para satisfazer um contrato, um padrão ou qualquer outra formalidade imposta pela documentação."

Um requisito pode, para ser melhor definido, ser classificado: Abran et al. (2004), assim como outros autores (B. Nuseibeh, 2007), (Sommerville, 2011), (Widrig e Leffingwell, 1999) para citar alguns, classificam os requisitos em requisitos funcionais e requisitos não-funcionais. Um requisito funcional descreve as funcionalidades do software, enquanto que um requisito não-funcional são aqueles que impõem restrições à solução a ser desenvolvida (Abran et al., 2004). Geralmente, os requisitos não-funcionais podem ser classificados em sub-categorias. Estas categorias podem ser: performance, manutenibilidade, segurança, confiabilidade, interoperabilidade, ou outras.

Há diversas formas de representar requisitos, em textos utilizando linguagem natural (Widrig e Leffingwell, 1999), utilizando linguagens de especificação de requisitos como as que serão apresentadas no Capítulo 3 ou podem ser representadas em forma de História de usuário (McDonald, 2016), como será descrito na Seção 2.3.

2.3 HISTÓRIA DE USUÁRIO

A História de Usuário surgiu com o método ágil *extreme programming (XP)* em 1998 (McDonald, 2016). Elas surgiram para serem utilizadas como uma das peças do principal processo de planejamento do XP, o *planning game*, já que elas definem o que o usuário quer que seja feito. No início, as histórias de usuário eram textos curtos e não estruturados (David Astels, 2002), então em 2001 o formato *Role-feature-reason* surgiu na empresa Connextra, por este motivo este formato também é conhecido como Formato Connextra (McDonald, 2016).

O Formato Connextra é:

As a [Stakeholder], **I want** [Feature] **So that** [Benefit].

Embora a estrutura anterior seja a mais difundida, há ainda uma variante baseada nela:

In order to [Benefit], **a** [Stakeholder] **Wants to** [feature].

O Formato Connextra, permite a identificação de três importantes itens de um requisito:

1. Para quem este requisito deve ser criado?

2. O que deve ser criado?

[5] `https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/Requisito/` - Acessado em 13/12/2018

3. Por que deve ser criado?

O *"Para quem?"* serve para guiar o desenvolvedor na forma que ele exibirá os dados e quais dados serão exibidos. O *"O que?"* expõe o requisito que deverá ser atendido. Por fim o *"Por que?"* tem como objetivo mostrar o valor de negócio que o *stakeholder* percebe neste requisito (Helm e Wildt, 2014). Um exemplo de História de Usuário no Formato *Connextra* é mostrada na Figura 2.2.

```
Feature: Account Holder withdraws cash
    As an account holder,
    I want to withdraw cash from an ATM,
    So that I can get money when the bank is closed.
```

Figura 2.2: Exemplo de História de Usuário no Formato Connextra em inglês.

Ainda em 2001 uma fórmula para Histórias de Usuário foi proposta, conhecida como *The Three C's* (Cartão, Conversa, Confirmação). Foi introduzida por Ron Jeffries com o objetivo de diferenciar História de Usuário 'sociais' de requisitos 'documentários', como os casos de uso (McDonald, 2016).

Nesta fórmula, o primeiro 'C' se refere a um Cartão (ou usualmente uma nota de *Post-It*). O cartão é um objeto físico que dá forma tangível e duradoura ao que, de outra forma, seria apenas uma abstração.

O segundo 'C' representa a conversa que ocorre em diferentes horários e lugares durante um projeto entre as várias pessoas envolvidas por uma determinada característica de um produto de software: clientes, usuários, desenvolvedores, testadores; esta conversa é em grande parte verbal, mas a maioria é complementada por documentação;

Por fim, o terceiro 'C'. Ele representa a confirmação das conquistas dos objetivos conversados. Portanto, quanto mais formal melhor.

Dois anos mais tarde, em 2003, o acrônimo INVEST foi criado, por Bill Wake, para definir um conjunto de critérios que devem ser seguidos para assegurar a qualidade das Histórias de Usuário (McDonald, 2016). O acrônimo siginifica (Helm e Wildt, 2014):

- ***I*** *ndependent* - Uma história deve ser independente de outras histórias
- ***N*** *egotiable* - Deve ser possível negociar a História de Usuário com o cliente
- ***V*** *aluable* - A História deve agregar valor ao cliente
- ***E*** *stimable* - A História deve ser estimável
- ***S*** *mall* - A História deve ser pequena
- ***T*** *estable* - A História deve ser testável

Em 2006, ferramentas de teste de aceitação como o *Cucumber*, *RSpec* e o *JBehave* introduziram uma forma de implementar testes com Histórias de Usuários (McDonald, 2016). Para tanto o formato *Given – When – Then* surgiu. Basicamente, neste formato, cenários são descritos utilizando marcadores de estado atual (Given), gatilhos de mudança de estado (When) e o estado novo (Then). A Figura 2.3 exemplifica o uso das palavras *Given – When – Then*.

Por ser um formato descritivo que utiliza linguagem natural, o padrão de História de Usuário *Given – When – Then* começou a ser utilizado na técnica de desenvolvimento de software *Behavior Driven Development* (BDD) que será explicada na Seção 2.4.

```
 1: Feature: Some terse yet descriptive text of what is desired
 2:   Textual description of the business value of this feature
 3:   Business rules that govern the scope of the feature
 4:   Any additional information that will make the feature easier to understand
 5:
 6:   Scenario: Some determinable business situation
 7:     Given some precondition
 8:       And some other precondition
 9:     When some action by the actor
10:       And some other action
11:       And yet another action
12:     Then some testable outcome is achieved
13:       And something else we can check happens too
14:
15:   Scenario: A different situation
16:       ...
```

Figura 2.3: Arquivo escrito utilizando o formato *Given–When–Then* implementado pelo Gherkin

2.4 BEHAVIOR DRIVEN DEVELOPMENT (BDD)

Segundo Fox (2016), Behavior Driven Development (BDD) é uma técnica de desenvolvimento de software que surgiu do Test Driven Developement (TDD) na primeira década dos anos 2000. O BDD utiliza linguagens de domínio específico (DSL) para transformar frases em testes executáveis.

Nesta técnica, o desenvolvedor cria o teste de aceitação do recurso que implementará, depois cria o novo recurso. Desta forma ele irá criar exatamente o recurso requisitado pelo cliente. Este processo, também conhecido como documentação viva, por ser executável, ou por especificação por exemplo (Adzic, 2011), é utilizado por equipes de desenvolvimento ágil que almejam automatizar os testes de aceitação de seus produtos.

Como o método de desenvolvimento é incremental, os recursos entregues no início do desenvolvimento terão que ser testados mais de uma vez. Para resolver este problema, ferramentas que automatizam testes são essenciais para garantir a integridade do software.

Estas ferramentas existem para diversos tipos de técnicas de desenvolvimento de software. No caso do BDD, algumas das ferramentas existentes são Cucumber, RSpec e o JBehave. Elas permitem que os desenvolvedores criem seus testes e os executem para validar se todas as funcionalidades implementadas do software estão funcionando sem problemas.

2.5 *CUCUMBER*

O *Cucumber* é uma ferramenta de teste de aceitação. Ele foi implementado para diversas linguagens de programação como o Ruby, Java, Javascript e outras (Aslak Hellesøy, 2008). Sua implementação utiliza a linguagem *Gherkin* para descrever os cenários que deverão ser testados.

Para trabalhar com *Cucumber*, o desenvolvedor escreve um código, na linguagem suportada que preferir, que executará uma linha escrita em *Gherekin*, ou seja, uma linha de uma História de Usuário. Com a execução desta linha virá o resultado do teste: Falho, não implementado ou Passou. Caso o resultado seja não implementado o desenvolvedor precisa definir o que deverá ser executado nas linhas não implementadas.

Caso o teste retorne falho, duas coisas podem ter acontecido: Ou o desenvolvedor programou o teste com erro, ou o teste encontrou falhas no software que deverão ser analisadas. Entretanto, se o programador codificou corretamente o teste e ele retornar 'passou', significa que o teste encontrou exatamente o que o desenvolvedor esperava e portanto a funcionalidade está correta

2.6 CONCLUSÃO

Neste Capítulo foram apresentados alguns conceitos utilizados durante o desenvolvimento deste trabalho. Inicialmente foi apresentado o conceito de *Stakeholder*. Nas definição fornecida, foi especificado o público alvo deste trabalho, os usuários não técnicos. Em seguida foi definido o que são requisitos. Eles fazem uma interface entre diferente tipos de *stakeholders*, clientes e desenvolvedores (Widrig e Leffingwell, 1999). Em seguida foi apresentada uma das formas de representar os requisitos, a História de Usuário. Uma técnica de desenvolvimento foi apresentada em seguida, junto com uma de suas ferramentas que utiliza História de Usuários.

No próximo Capítulo serão expostas alternativas de representação de requisitos, através de linguagens de especificação de requisitos. As linguagens apresentadas foram encontradas após uma revisão da literatura.

3 TRABALHOS RELACIONADOS

Neste Capítulo, serão apresentadas linguagens de especificação de requisitos já criadas. As linguagens de especificação aqui descritas foram coletadas através de uma revisão de literatura em quatro bibliotecas digitais diferentes: IEEEXplore, ACM, ScienceDirect e SpringerLink. Então, foram utilizadas as strings de busca que estão na Figura 3.1.

Inicialmente, os artigos encontrados foram filtrados por ano: Os artigos mais antigos que 2012 não foram utilizados. O ano 2012 foi escolhido porque ele marca cinco anos de pesquisas entre o início desta pesquisa e os trabalhos anteriores. Na segunda filtragem, dos artigos restantes, foram selecionados os artigos cujos títulos aparentavam definir ou comparar linguagens de especificação de requisitos. Por fim, dos artigos selecionados na segunda filtragem foram lidas as seções de resumo e conclusão. A leitura destas duas seções ajudou o autor deste trabalho a identificar os artigos se enquadrassem em uma ou mais das situações:

1. Definir uma linguagem de especificação
2. Definir um método de utilização de uma linguagem de especificação de requisitos
3. Comparar duas ou mais linguagens de especificação de requisitos

Os trabalhos selecionados, que podem ser vistos na Tabela 3.1, compuseram os artigos finais que foram utilizados como base para o desenvolvimento deste trabalho. O Número de artigos em cada fase da revisão de literatura pode ser vista na tabela da Figura 3.1.

String de busca	Biblioteca	Número de artigos			
		Sem filtro	Depois do primeiro filtro	Depois do segundo filtro	Depois do terceiro filtro
Requirement "Specification Language" OR Requisit "Specification Language"	IEEEXplore	1901	303	30	12
(+requirement +specification +language) AND acmdlTitle:(+language)	ACM	1607	553	24	9
(requirement specification language) and TITLE(requirement language)	ScienceDirect	34	18	14	7
requirement AND specification AND language'	SpringerLink	191	80	22	2

Figura 3.1: Metodologia para revisão de literatura

A revisão foi realizadas em dois períodos diferentes: O primeiro foi em outubro de 2017, quando os dados da tabela da Figura 3.1 foram coletados. O segundo período foi no início de dezembro de 2018, nesta ocasião apenas dois trabalhos novos ((Serna et al., 2017) e (Haidrar et al., 2017)) atenderam os filtros do autor e não estão na Tabela 3.1.

A seguir as linguagens encontradas nesta revisão de literatura serão apresentada. A ordem de apresentação das linguagens não segue nenhum critério especial, com exceção das linguagens Alloy (Seção 3.5) e linguagem DASH 3.6, já que, como será apresentado, a linguagem DASH é uma extensão da linguagem Alloy.

Tabela 3.1: Artigos selecionados na revisão de literatura

Base	Artigo	Autor
IEEEXplore	Alneelain: A formal specification language	(Ali et al., 2017)
IEEEXplore	ReSA: An ontology-based requirement specification language tailored to automotive systems	(Mahmud et al., 2015)
IEEEXplore	Preliminary Experience Using JetBrains MPS to Implement a Requirements Specification Language	(Savić et al., 2014)
IEEEXplore	The Concepts and Ontology of SiSL: A Situation-Centric Specification Language	(Xie et al., 2012)
IEEEXplore	Semantic annotation of a formal grammar by SemanticPatterns	(Schraps e Peters, 2014)
IEEEXplore	RSL-IL: An interlingua for formally documenting requirements	(De Almeida Ferreira e Da Silva, 2013)
IEEEXplore	RSLingo: An information extraction approach toward formal requirements specifications	(De et al., 2012)
IEEEXplore	Facilitating transition from requirements to code with the ReDSeeDS tool	(Smialek e Straszak, 2012)
IEEEXplore	Model to specify real time system using Z and Alloy languages: A comparative approach	(Dwivedi e Rath, 2012)
IEEEXplore	SOFIA: An Algebraic Specification Language for Developing Services	(Liu et al., 2014)
IEEEXplore	A formal specification language for modeling agent systems	(Subburaj e Urban, 2013)
IEEEXplore	VDMPad: A Lightweight IDE for Exploratory VDM-SL Specification	(Oda et al., 2015)
ACM	A Textual Domain Specific Language for user interface Modelling	(Karu, 2013)
ACM	A Pattern Language for Use Cases Specification	(da Silva et al., 2015)
ACM	Solving the Bank with Rebel: On the Design of the Rebel Specification Language and Its Application Inside a Bank	(Stoel et al., 2016)
ACM	Towards a Behavior-oriented Specification and Testing Language for Multimodal Applications	(Hesenius et al., 2014)
ACM	Specifying Safety Requirements with GORE Languages	(Vilela et al., 2017)
ACM	A Comparative Evaluation of the Z, CSP, RSL, and VDM Languages	(Nami e Hassani, 2009)
ACM	A Comparative Study of Two Formal Specification Languages: Z-notation & B-method	(Kaur et al., 2012)
ACM	Program Synthesis Using Natural Language	(Desai et al., 2015)
ACM	A Restricted Natural Language Based Use Case Modeling Methodology for Real-time Systems	(Zhang et al., 2017)
ScienceDirect	A semi-automated approach for generating natural language requirements documents based on business process models	(Aysolmaz et al., 2018)
ScienceDirect	Functional grouping of natural language requirements for assistance in architectural software design	(Casamayor et al., 2012)
ScienceDirect	Enriching UsiXML language to support awareness requirements	(Figueroa-Martinez et al., 2013)
ScienceDirect	A language for multi-perspective goal modelling: Challenges, requirements and solutions	(Overbeek et al., 2015)
ScienceDirect	Conceptual modeling of natural language functional requirements	(Vidya Sagar e Abirami, 2014)
ScienceDirect	Analyzing the understandability of Requirements Engineering languages for CSCW systems: A family of experiments	(Teruel et al., 2012)
ScienceDirect	PLANT: A pattern language for transforming scenarios into requirements models	(Wang et al., 2013)
SpringerLink	Running Out of Words: How Similar User Stories Can Help to Elaborate Individual Natural Language Requirement Descriptions	(Bäumer e Geierhos, 2016)
SpringerLink	Fuzzy Representation for Flexible Requirement Satisfaction	(Anggraini e Martin, 2018)

3.1 GHERKIN

O Gherkin foi criado em 2008 por Aslak Hellesøy (Hellesøy, 2018), e, segundo a documentação do GitHub (Bolic, 2017), é uma linguagem *Business Readable, Domain Specific Language*. Isso permite com que stakeholders que não são desenvolvedores consigam utilizar a linguagem. Portanto a linguagem Gherkin permite que o comportamento do software seja descrito sem detalhar como ele será implementado em qualquer um dos sessenta idiomas disponíveis.

Na prática o Gherkin tem dois propósitos: Documentar e automatizar testes, com ferramentas como o Cucumber, descrito na Seção 2.5. A gramática do Gherkin deriva das Histórias de Usuário do estilo *Given – When – Then*, e portanto ela pode acabar sendo utilizada como única forma de formalizar requisitos. Contudo, o formato de História de Usuário *Given – When – Then* documenta os possíveis fluxos do sistema, faltando especificação de detalhes como: quem é o ator, o que deve ser feito e por que deve ser feito. Estes atributos são obtidos com o uso da Histórias de Usuário que utilizam o padrão Connextra.

As palavras chaves do Gherkin são: *Feature*, *Scenario*, *Background*, *Scenario Outline* e *Examples*. A palavra *Feature* é utilizada para documentar o nome do recurso que será descrito. A palavra *Scenario* é a palavra chave usada para criar a descrição de um cenário. Após esta palavra chave vem o nome do cenário, que identifica o que será descrito. *Background* é usado para descrever situações de pré-requisito/contexto para todos os cenários do arquivo de descrição. *Scenario Outline* permite a utilização de variáveis nos cenários para que exemplos sejam utilizados para testar o recurso descrito. Estes exemplos são descritos em formato de tabela, depois de definir o cenário sob a palavra chave *Examples*. Os cenários são descritos utilizando o padrão de História de Usuário *Given – When – Then*.

3.2 ALNEELAIN

A linguagem Alneelain (Ali et al., 2017) foi criada para ser uma linguagem formal sem requisitar muitos detalhes matemáticos de seu usuário. Os autores afirmam ainda que esta linguagem é precisa, clara e concisa mesmo simplificando a parte matemática da especificação. A figura 3.2 mostra o exemplo da especificação de uma estrutura de dados 'pilha' utilizando a linguagem Alneelain.

Mesmo suprimindo muitos detalhes da matemática o usuários da linguagem Alneelain precisam ter conhecimentos técnicos em lógica e matemática para conseguir especificar e interpretar sistemas com esta linguagem.

3.3 RESA

ReSA é uma linguagem de especificação de requisitos baseada em ontologia otimizada para sistemas automotivos (Mahmud et al., 2015). Por ser uma linguagem baseada em ontologia, para utilizá-la, é preciso escrever estruturas condicionais, assemelhando-se muito a uma linguagem de programação, como mostra a figura 3.3.

O requisito 1 do sistema, representado na figura 3.3 utilizando a linguagem ReSA é: *"Dado um Limitador ajustável de velocidade (Adjustable Speed Limit - ASL), se o ASL estiver ativo e o motorista pressionar o botão PAUSAR ou o veículo sair do modo ligado, as funções do ASL param para limitar a velocidade do veículo. A indicação do status do ASL é trocada de ativo para habilitado. O ASL muda para o estado habilitado."*.

O ReSA é uma linguagem criada para um domínio específico, especificação de sistemas automotivos, logo é esperável que ela exija de seus usuários o domínio do ambiente em que ela

```
specification Stack;
type
      itemtype : char;
input
      vop top: itemtype ,
      vop size: integer ,
      vop empty: Boolean
      oop init, pop, push(itemtype )
endinput;
output
      itemtype ∧ error ∧ Boolean ∧ integer
endoutput;
variable
      a: itemtype ,
      h: inputstar ,
      hprime: inputstar ,
      hplus: inputplus ;
axioms
axiom topAxioms:
      Stack(init.top) = error &
      Stack(init.h.push(a).top) = a ,

axiom sizeAxiom:
      Stack(init.size) = 0,
axiom emptyAxioms:
      Stack(init.empty)=true &

      Stack(init.push(a).empty)=false
endaxioms;
rules
rule initRule:
      Stack(h.init.hprime)=Stack(init.hprime) ,
rule initpopRule:
      Stack(init.pop.h) = Stack(init.h) ,
rule pushpopRule:
      Stack(init.h.push(a).pop.hplus) = Stack(init.h.hplus) ,
rule sizeRule:
      Stack(init.h.push(a).size) =1+ Stack(init.h.size) ,
rule emptyRules:
      Stack(init.h.push(a).hprime.empty)=>
      Stack(init.h.hprime.empty)&
      Stack(init.h.empty) => Stack(init.h.pop.empty) ,
rule VopRules:
      Stack(init.h.top.hplus)=Stack(init.h.hplus) &
      Stack(init.h.size.hplus)=Stack(init.h.hplus) &
      Stack(init.h.empty.hplus)=Stack(init.h.hplus)
```

Figura 3.2: Especificação de pilha em Alneelain Ali et al. (2017)

será utilizada. Por este mesmo motivo, é natural que esta não seja uma linguagem que permita a comunicação simples e fácil entre diferentes tipos de stakeholders.

3.4 NOTAÇÃO Z

A notação Z é baseada em teoria de conjuntos e lógica matemática (Davies e Woodcock, 1996). A notação Z agrupa objetos matemáticos em esquemas: padrões de declaração e limitações.

ReSA_Req 1 : Rewriting Req1

ReqV: ▷ Vehicle-level

```
IF (<ASL:VF> is <activated:State> AND
(<Driver> <presses:ActOnInDev> <PAUSE
button:InDevice> OR
   vehicle is not in <running mode:Mode))
THEN
  <ASL:VF> shall <stop to limit:ActOnPara>
<vehicle speed:Stimuli> AND
  <ASL status:Response> shall be <set
to:ActOnPara> <enabled:State> AND
  <ASL:VF> shall be <enabled:State>
ENDIF
```

Figura 3.3: Especificação de sistema escrita com ReSA (Mahmud et al., 2015)

Além disso, a linguagem Z utiliza tipos. Um tipo na notação Z é um conjunto maximal. Como a notação Z utiliza tipos, é possível verificar programas que implementam a especificação escrita com a notação Z.

A figura 3.4 exemplifica o uso da notação Z na definição de um módulo (*timeout*) de um escalonador de processos de um sistema operacional

ATimeOut

$\Delta AScheduler$

$p! : PId$

$current \neq nullPId$

$current' = nullPId$

$ready' = ready \cup \{current\}$

$blocked' = blocked$

$free' = free$

$p! = current$

Figura 3.4: Especificação parcial de um sistema utilizando Notação Z (Davies e Woodcock, 1996)

O que a figura 3.4 especifica é que, em um escalonador de processos, a cada intervalo de tempo um processo utiliza o processador e o que estava utilizando o processador é colocado novamente na lista de processos prontos para serem executados.

Assim como acontece com as linguagens anteriores, a notação Z não é muito indicada para apresentar para usuários que não tenham conhecimento de teoria de conjuntos e lógica matemática.

3.5 ALLOY

Alloy é uma linguagem para descrição inspirada pela notação Z. É uma linguagem baseada em lógica e relacionamentos. A criação da sintaxe da linguagem Alloy foi influenciada por linguagens de modelagem (Daniel Jackson, 2017). A figura 3.5 exemplifica o uso do Alloy.

A figura 3.5, Dwivedi e Rath (2012) criaram uma definição de uma máquina de atendimento automático. No exemplo são especificados a inserção de cartão na máquina e a falta de dinheiro para troco.

3.6 DASH

DASH, acrônimo de *Declarative Abstract State Hierarchy*, é uma camada para a linguagem Alloy que foi criada por Serna et al. (2017). Esta linguagem descreve um conjunto

```
module ATMSystem
sig Identifier {}
abstract sig ATM_STATE {}
one sig Start_Trans , ATMWaitCard , ATMWaitPin ,
ATMWaitInst , Rem_Cash ,  Rem_Card extends
ATM_STATE {}
abstract sig Operation {}
one sig Enter_Card, Enter_Pin, CASH extends Operation{}
sig ATM {
            a_card : lone Identifier ,
            a_pin : lone Identifier ,
           a_state : one ATM_STATE ,
            bal : Identifier -> one Int ,
            op : lone Operation
}
pred nocashchange  [a, a': ATM] {
a'.a_card=a.a_card && a'.a_pin = a.a_pin
}
pred entercard [a, a': ATM, c: Identifier ] {
(a.a_state = ATMWaitCard or a.a_state = Start_Trans ) &&
(a'.a_card = c)  && (a'.a_pin = a.a_pin ) &&
Nocashchange [a, a'] && (a.a_state=Start_Trans => a'.a_state
=ATMWaitPin)
}
```

Figura 3.5: Especificação de uma máquina de atendimento automático simplificado em Alloy (Dwivedi e Rath, 2012)

de transições de estados. Cada estado é descrito utilizando a linguagem Alloy, enquanto que cada transição é descrita utilizando a gramática do DASH, já que a linguagem Alloy não suporta, explicitamente, a descrição de modelos dinâmicos/comportamentais (Serna et al., 2017) . O exemplo da Figura 3.6 mostra a descrição de um sistema de aquecimento escrito em DASH.

No exemplo da Figura 3.6, é possível perceber que para utilizar o DASH, é preciso ter mais familiaridade em especificação de sistemas e programação, visto que parte da sintaxe é similar à sintaxe encontrada em linguagens de programação. Então, como o Alloy, esta não é uma linguagem voltada a comunicação entre diversos tipos de stakeholders.

3.7 MÉTODO B

O Método B é uma linguagem de especificação que utiliza lógica, teoria de conjuntos e uma linguagem de substituição generalizada para descrever sistemas (Attiogbé, 2018). Assim como as outras linguagens apresentadas até esta Subseção, ela é uma linguagem que requer conhecimentos matemáticos mais avançados e não é ideal para apresenta a um stakeholder. A figura 3.7 mostra a especificação de um algoritmo de ordenação de números naturais.

Como pode ser observado na figura 3.7, além de exigir muitos conhecimentos específicos, os elementos léxicos que compõem a linguagem não ajudam muito as pessoas que não têm conhecimentos mais avançados em ciências exatas a entender o que cada seção da especificação tem como objetivo.

3.8 VDM

A linguagem Vienna Development Method (VDM) é um método formal orientado por modelo baseado em matemática discreta e lógica. Este método consegue suportar a descrição de

```
abstract sig ValvePosition {}
abstract sig Room {}
...

conc state HeatingSystem {
  valvePosition: Room -> ValvePosition
  desiredTemp: Room -> Int
  actualTemp: Room -> Int
  occupied: Room
  requestHeat: Room

  event activate {}
  event deactivate {}

  action adjValve [
    all r:occupied |
      r.actualTemp < r.desiredTemp =>
      r.valvePosition' = OpenPosition
  ] {}

  condition roomsNeedHeat [
    some requestHeat
  ] {}

  init {
    all r: Room |
      r.valvePosition = ClosedPosition
  }

  conc state Controller {
    default state Off { }
    state Error { }
    state On {
      default state Idle{
        trans t1 {
          when roomsNeedHeat
          goto HeaterActive
          send activate
        }
        trans t2 {
          from HeaterActive
          when (not roomsNeedHeat)
          send deactivate
        }
      }
      state HeaterActive{
        default state ActivatingHeater{}
        state HeaterRunning{}
      }

      trans t3 {
        on heatSwitchOff
        goto Off send deactivate
      }
      trans t4 {on furnaceFault goto Error}
    }
  }

  conc state Bedroom {
    default state NoHeatRequestested {}
    state HeatRequested {
      default state  IdleHeating{}
      state WaitForCool{
        trans t5 {on waitedForCool do adjValve}
      }
    }
  }
}
```

Figura 3.6: Exemplo de especificação de um sistema de aquecimento em DASH (Serna et al., 2017)

```
MACHINE /* Specify the sorting of a set of naturals */
        Sort
CONSTANTS
        sortOf /* defining a function */
PROPERTIES
   sortOf : FIN(NAT) +-> seq(NAT) &
     %ss.(ss : FIN(NAT) =>
     (ran(sortOf(ss)) = ss &
      %(ii,jj).(ii : dom(sortOf(ss)) &    jj : dom(sortOf(ss)) &
      ii < jj   =>    (sortOf(ss))(ii) < (sortOf(ss))(jj) )
    ) )
END
```

Figura 3.7: Especificação de um algoritmo de ordenação de números naturais (Attiogbé, 2018)

dados e de funcionalidades (Oda et al., 2015). O exemplo da figura 3.8 mostra a especificação de um gerenciador de estoque de uma loja de bebidas.

A figura 3.8 mostra claramente que a estrutura de uma especificação em VDM é muito similar a uma estrutura de linguagem de programação. Este fato faz com que esta linguagem não seja a mais apropriada para apresentar à stakeholders, já que é preciso ter muitos conhecimentos técnicos prévios para conseguir interpretar corretamente a especificação escrita com VDM.

3.9 I*

i* é uma linguagem de modelagem adequada para as fases iniciais da modelagem do sistema, a qual auxilia no entendimento do problema. Sua especificação prevê dois tipos principais de modelos: O modelo de Dependência Estratégica e o modelo de Razão Lógica (Yu, 2009) .

```
types
Item = <BEER> | <WINE>;
Bag = map Item to nat1;
Order = seq of (Item * nat1);
Money = real;

values
empty :Bag = { |-> };

values
pricelistForPurchase
  ={<BEER> |-> 0.5, <WINE> |-> 12};
pricelistForSales
  = {<BEER> |-> 1, <WINE> |-> 20};

state Inventory of
stock : Bag
balance : Money
init s == s = mk_Inventory(empty, 0)
inv mk_Inventory(s, b) == b >= 0
end

operations
Buy : Order ==> ()
Buy(order) == for mk_(item, num) in order do
  let
    current_num =
      if item in set  dom stock
      then stock(item)
      else 0,
    new_balance =
      balance - pricelistForPurchase(item)*num
  in
    if new_balance >= 0
    then (
      stock := stock++{item |-> current_num+num};
      balance := new_balance )
    else skip;
Sell : Order ==> ()
Sell(order) == for mk_(item, num) in order do
  let
    new_num = stock(item) - num
  in
    (if new_num > 0
     then stock := stock++{item|->new_num}
     else stock := {item} <-: stock;
    balance:=balance+pricelistForSales(item)*num
    );
```

Figura 3.8: Especificação de um gerenciador de estoque de uma loja de bebidas com VDM (Oda et al., 2015)

Segundo Yu (2009), o modelo de Dependência Estratégica é uma rede de dependência direcionada entre atores que permite especificar o funcionamento do sistema, sem muitos detalhes, para que uma análise do sistema seja simplificada. Um exemplo deste modelo é mostrado na figura 3.9, onde é especificado um sistema de saúde simplificado.

O modelo de Razão lógica, mostrado na figura 3.10, permite modelar tarefas, metas, recursos e submetas para cada ator. Desta forma é possível planejar como alcançar cada meta e quais tarefas cada ator deve executar. De uma maneira geral, o modelo de Razão lógica é similar ao modelo de Dependência Estratégica. Entretanto, o diagrama de Razão Lógica é mais detalhado, o que permite uma análise mais precisa do problema.

O mesmo sistema de saúde que foi modelado com o diagrama de Dependência Estratégica na Figura 3.9 encontra-se modelado utilizando o diagrama de Razão Lógica na Figura 3.10.

Diferentemente das linguagens apresentadas até então, o i* é uma linguagem gráfica. Por este motivo esta linguagem tem mais chances de ser usada com clientes e usuários, mesmo

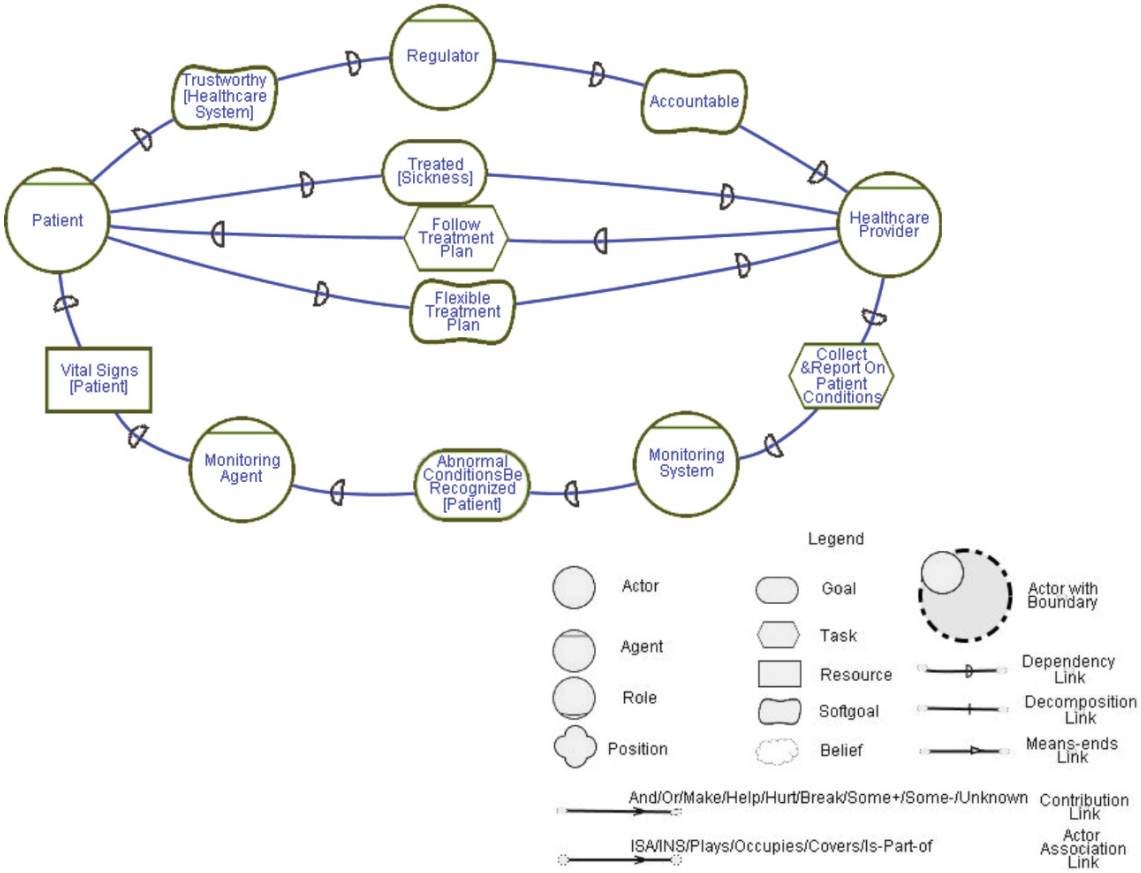


Figura 3.9: Especificação de um sistema de saúde usando o modelo de Dependência Estratégica do i* (Yu, 2009)

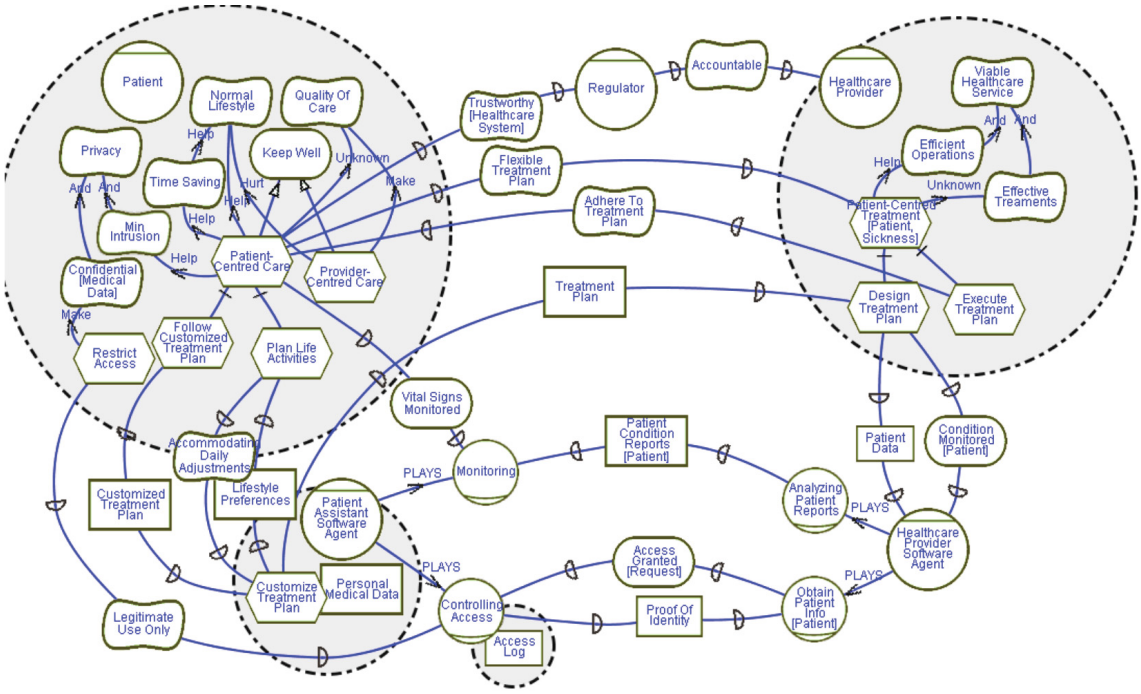


Figura 3.10: Especificação de um sistema de saúde usando o modelo de Razão Lógica do i* (Yu, 2009)

não sendo o ideal. Esta possibilidade é obtida pelo fato que os conceitos necessários para o entendimento do diagrama podem ser explicados de forma simples. O maior problema é a especificação de sistemas muito grandes, onde a quantidade de ligações e elementos visuais

podem ser um empecilho para o entendimento, já que podem aumentar a complexidade cognitiva (Rauterberg, 1996).

Outro fator que deve-se considerar como problema é o fato de que provavelmente um stakeholder não criará um diagrama destes, pois a criação de um diagrama é uma ação mais complexa do que entender um diagrama pronto, considerando que a notação, através de formas diferentes, requer precisão para a correta especificação do sistema desejado.

3.10 KAOS

KAOS é uma técnica orientada a metas, assim como o i*, para especificar requisitos (Dardenne et al., 1993). O KAOS possui quatro modelos: Modelo de metas, Modelo de responsabilidade, Modelo de operação e Modelo objeto.

O modelo de metas tem como objetivo elencar os requisitos do sistema, seus agentes e o relacionamento entre eles. A figura 3.11 exemplifica como é um modelo de metas do KAOS (Dardenne et al., 1993)

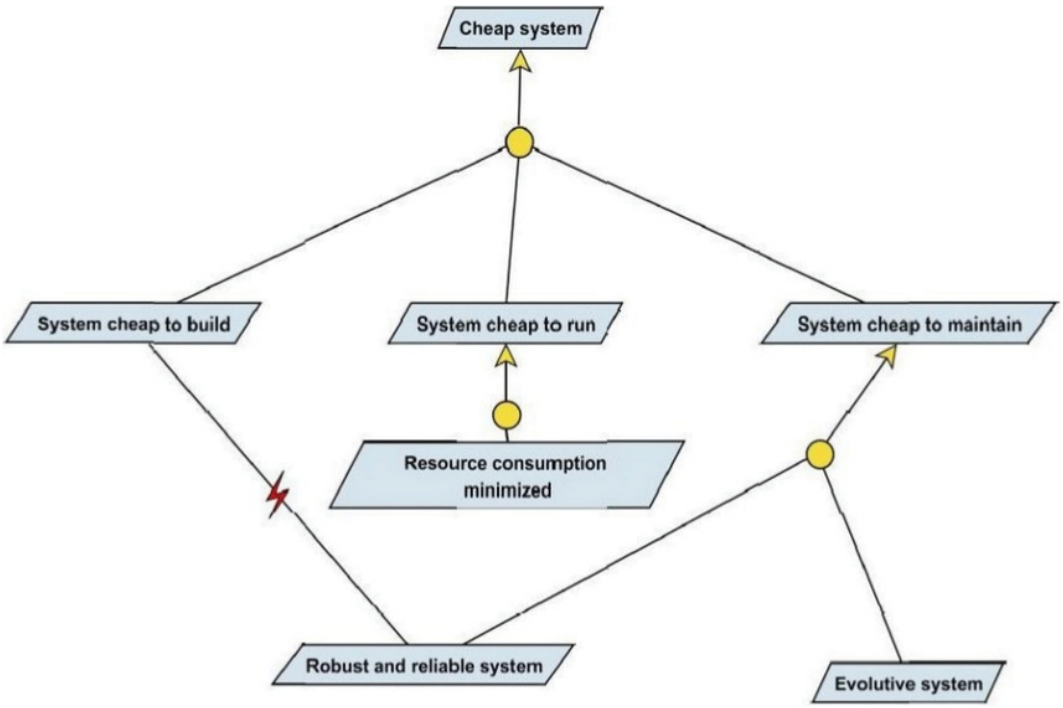


Figura 3.11: Especificação de alguns requisitos não funcionais de um elevador utilizando o modelo de metas do KAOS (Respect-IT, 2007)

O modelo objeto é usado para definir e documentar os conceitos do domínio da aplicação relevantes em relação aos requisitos conhecidos. A figura 3.12 exemplifica como é um Modelo objeto do KAOS.

O Modelo objeto do KAOS é compatível com o diagrama de classes da UML (Unified Model Language), onde as entidades do KAOS correspondem a classes UML e as associações do KAOS correspondem a associações da UML. Além disso, herança está disponível para todos os tipos de objeto, inclusive associações (Respect-IT, 2007).

O terceiro modelo do KAOS, Modelo de operação, descreve todos os comportamentos que os agentes precisam para executar seus requisitos. A figura 3.13 mostra como pode ser modelado o controle de um elevador.

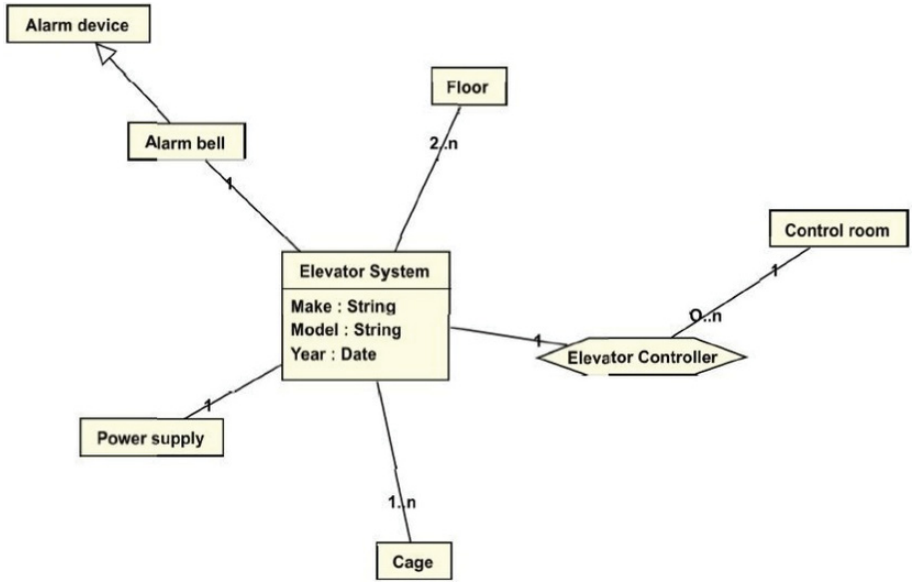


Figura 3.12: Especificação de um elevador utilizando o modelo objeto do KAOS (Respect-IT, 2007)

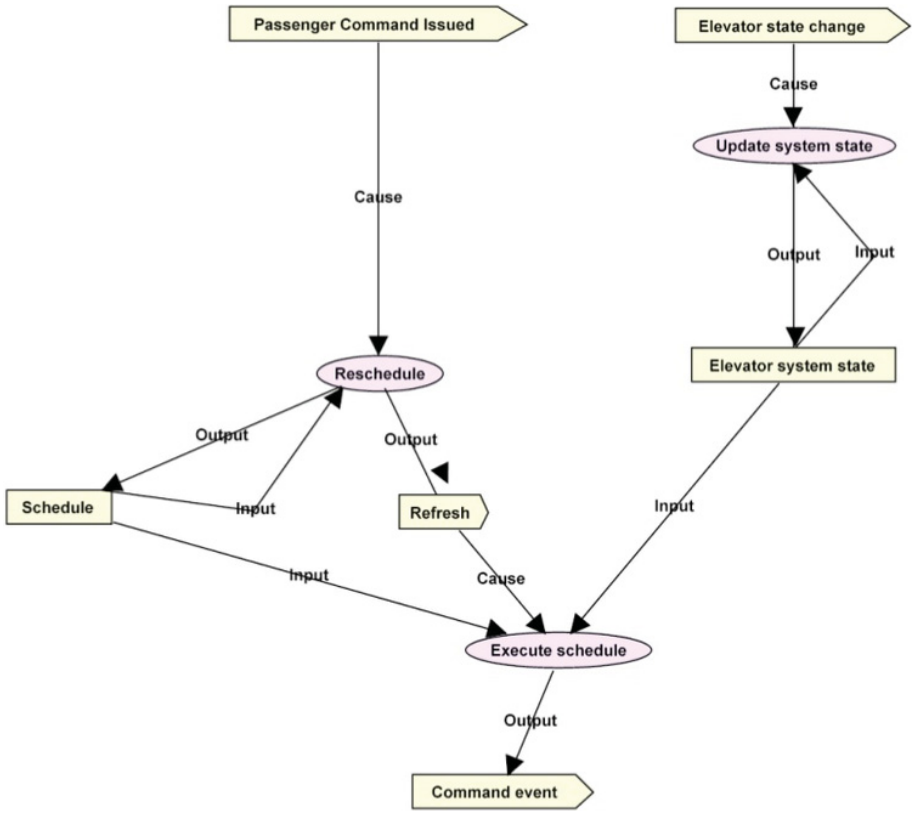


Figura 3.13: Especificação do controle do elevador utilizando o modelo de operação do KAOS (Respect-IT, 2007)

Por fim, o Modelo de responsabilidade é o modelo que contém todos os diagramas de responsabilidade. Um diagrama de responsabilidade descreve, para cada agente, os requisitos e expectativas pelos quais ele é responsável. Por definição, o Modelo de responsabilidade é derivado do Modelo de metas, como pode ser observado na figura 3.14

O KAOS é um conjunto de modelos para especificar requisitos. Estes modelos podem ser representados graficamente, como mostrado ao longo desta Subseção. Sua utilização por stakeholders não é muito recomendada por possuir muitos elementos gráficos que precisam ser

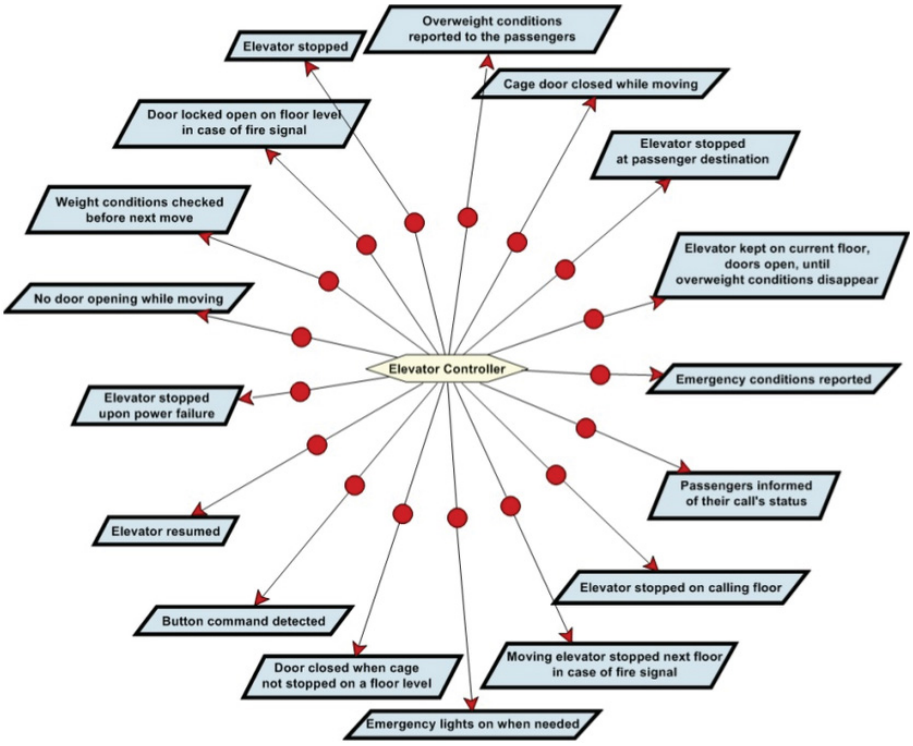


Figura 3.14: Especificação das responsabilidades do controlador do elevador utilizando o modelo de responsabilidade do KAOS (Respect-IT, 2007)

precisamente atribuídos. Além disso, um *stakeholder* pode não entender o significado de cada um dos quatro diagramas que são criados para especificar os requisitos do sistema. Outro ponto que vale o destaque é o esforço cognitivo considerável requisitado dos usuários iniciantes para conseguir unificar os quatro diagramas. Já que, segundo Rauterberg (1996), a complexidade cognitiva é baseada na complexidade do sistema e na complexidade da tarefa a ser executada, é possível esperar que os utilizadores de todos os diagramas do KAOS tenham, em distintos níveis, dificuldade cognitiva para unificar os diagramas mentalmente. Pelo trabalho de Respect-IT (2007) pode ser percebido que a interpretação conjunta destes diagramas é essencial para a compreensão precisa do sistema especificado,

3.11 REQDL

A linguagem ReqDL foi criada por Haidrar et al. (2017) para descrever requisitos ao mesmo tempo em que são coletadas informações para permitir a rastreabilidade dos requisitos. Um exemplo de uso desta linguagem, descrevendo um sistema de refrigeração de um automóvel é mostrado na Figura 3.15.

Com o exemplo da Figura 3.15, é possível perceber que o ReqDL tem uma gramática um pouco mais próxima da gramática das linguagens naturais, permitindo que pessoas que não tenham estudado sobre a gramática, mas que possuem uma formação técnica, consigam entender o que foi especificado. Desta forma mais stakeholders são capazes de compreender a linguagem, mas não todos. Outro aspecto perceptível nesta linguagem é o cuidado que os autores tiveram em garantir a rastreabilidade da linguagem. É possível, só olhando o exemplo da Figura 3.15, perceber quase todas as estruturas são responsáveis por garantir a rastreabilidade dos requisitos.

```
ModelElement Block Radiateur
ModelElement Block Thermostat
ModelElement Block TemperatureSensor
ModelElement Testcase OptimalTemperature

Traceable Requirement TemperatureManagement
{ identifier:R0
type: General
level:StakeholderReq
source : 'RSD'
author : 'Client'
description:
subject 'Vehicle' shall 'Maintain the engine with optimal temperature'}

Traceable Requirement QuikTemp belongsTo TemperatureManagement
{ identifier: R01
type:  PerformanceRequirement
level:StakeholderReq
description: subject CCS should be able to "allow the engine to reach optimal temperature quickly"
the ModelElement OptimalTemperature}

Traceable Requirement HeatDecrease derivesFrom QuikTemp
{ identifier: R02
type: Functional
level:SystemReq
description: subject CCS should "allow the heat transfer from coolant to the air"
the ModelElement Radiator}

Traceable Requirement EngineCirculation derivesFrom QuikTemp
{ identifier: R03
type: Functional
level:SystemReq
description:
subject CCS should "restrict coolant flow to circulate through radiator"
the ModelElement Thermostat}

Requirement AlertManagement  {
identifier: R04
type: General
level: StakeholderReq
description:
subject CCS will "provide the user with the ability to supervise engine temperature "}

Requirement driverAlert refines AlertManagement {
identifier:R05
type: Functional
level: SystemReq
description: where the engine reaches the specified upper limit
subject CCS will provide the driver with Ability to be alerted
the ModelElement TemperatureSensor
```

Figura 3.15: Exemplo de especificação de um sistema de refrigeração automotiva em ReqDL (Haidrar et al., 2017)

3.12 RSLINGO

RSLingo é uma linguagem de extração de informações baseada em padrões linguísticos. Para tanto é utilizada a estratégia de multi-linguagens baseada em duas linguagens: RSL-PL (Pattern Language, inglês para linguagem de padrão) para definir os padrões linguísticos e RSL-IL (Intermediate Language, inglês para linguagem intermediária) para agir como a linguagem de especificação de requisitos (De Almeida Ferreira e Da Silva, 2013).

Com a ajuda destas duas linguagens, o RSLingo transforma texto de especificação de requisito escrito em linguagem natural para seu formato, o qual mescla a linguagem natural com suas próprias estruturas. Seu formato segue o exemplo da figura 3.16

O RSLingo é uma ferramenta que permite que usuários escrevam seus requisitos usando linguagem natural e transforma estes requisitos em um formato próprio. Esta abordagem facilita a demanda de requisitos, mas a comunicação entre stakeholders e desenvolvedores ainda continua como um problema, visto que apenas algumas partes da especificação original são processadas, já que processar linguagem natural ainda é difícil como notado por De et al. (2012). Desta forma pode ser que especificações ambíguas sejam mal interpretadas tanto pelo software, que utiliza um dicionário, quanto pelos desenvolvedores, gerando desta forma mal entendidos que as linguagens naturais podem causar.

3.13 CONCLUSÃO

Neste Capítulo foram apresentadas doze linguagens que especificam sistemas. Algumas delas, como Alloy, são utilizadas para a descrição de sistemas, outras como a ReqDL são usadas para a especificação de requisitos de sistemas.

As linguagens Gherkin, RSLingo e ReqDL são as linguagens que o autor desta dissertação acredita estar mais próximas de um usuário não técnico, exatamente pelo fato de que elas são as únicas linguagens, entre as linguagens citadas neste Capítulo, que utilizam linguagem natural

```
(PROJECT (...)
  (SYSTEM id:"sys-adsp" name:"AIDSPortugal" (...)
    (SYSTEM id:"sys-onln-expt" name:"Online Experts" (...)
      (BEHAVIOR
        (USE-CASES
          (USE-CASE id:"uc-ask-qstn" name:"ask a question"
                description:"The interaction through which an anonymous user
                is able to pose a question to a medical specialist."
            (ACCOMPLISHES goal:"gol-advc-qa")
            (RELATED-WITH functional:"FR-3.1.2.4.10")
            (ACTORS initiates:"act-anms-user" participates:"act-mdrr")
            (TRIGGERED-BY event:"evt-ask-qstn")
            (CONDITIONS
              (PRE (CONDITION usecase:"uc-nvgt-onln-exprt"))
              (POST
                (CONDITION entity:"ent-qstn" attribute:"publishing state"
                      current:"pending_approval")
                (CONDITION entity:"ent-qstn" attribute:"object state"
                      current:"active")))
            (REQUIRES entity:"ent-qstn")
          (SCENARIOS
            (SCENARIO name:"Default ask question procedure." type:@normal
                  sequential:"true"
              (STEP label:"1" text:"The 'anonymous user' selects the 'ask
                    question' functionality.")
              (STEP label:"2" text:"The system asks the 'anonymous user' to
                    insert: the 'question text' (mandatory), the 'author name'
                    (mandatory), and the 'author e-mail' (mandatory).")
              (STEP label:"3" text:"The 'anonymous user' fills the question
                    form's fields.")
              (STEP label:"4" text:"The system asks the 'anonymous user' to
                    select the 'medical specialist' to whom the question should
                    be directed.")
              (STEP label:"5" text:"The 'anonymous user' selects one of the
                    available 'medical specialists'.")
              (STEP label:"6" text:"The system challenges the 'anonymous
                    user' to provide evidence that he/she is a human being.")
              (STEP label:"7" text:"The 'anonymous user' meets the challenge.")
              (STEP label:"8" text:"The system validates the new question's
                    fields.")
              (STEP label:"9" text:"The system notifies the moderator.")
              (STEP label:"10" text:"The system displays a success message to
                    the 'anonymous user'."))
            (SCENARIO name:"Exceptions" type:@exception sequential:"true"
              (STEP label:"7.1a" text:"The 'anonymous user' fails the
                    challenge.")
              (CONTINUE label:"7.1b" next:"6" description:"The system
                    continues until the 'anonymous user' meets the
                    challenge.")))))))))
```

Figura 3.16: Exemplo de especificação transformada usando RSLingo (De Almeida Ferreira e Da Silva, 2013)

sem nenhuma estrutura gráfica, fato que tende a diminuir a complexidade cognitiva da linguagem, conforme discutido anteriormente na Seção 3.9.

Destas linguagens, o Gherkin é a linguagem que fornece a melhor legibilidade para usuários não técnicos, já que utiliza uma estrutura em conjunto com as linguagens naturais, que por forçar o uso de algumas palavras-chaves tenta evitar a ambiguidade das linguagens naturais. É verdade que o RSLingo e o ReqDL também permitem a escrita dos requisitos usando linguagem natural em suas próprias estruturas, entretanto as estruturas utilizadas pelo RSLingo não são tão legíveis quanto as do Gherkin. Já o ReqDL, tem como foco apenas a rastreabilidade dos requisitos, fazendo com que a linguagem, embora mais legível que o RSLingo, possa não

permitir um usuário não técnico entender a linguagem apenas lendo pela primeira vez, uma vez que que a forma de especificar os requisitos lembra um código-fonte.

Outro ponto que faz o Gherkin se destacar é fato dele ser usado para definir testes de aceitação. Então, quando utilizamos o Gherkin, além de especificar o requisito, especificamos seu funcionamento e definimos os testes que deverão ser programados. Esta característica permite a economia de tempo de desenvolvimento do software, pois ao mesmo tempo que são especificados os requisitos são, também, especificados os testes de aceitação. Desta forma as versões funcionais do software poderão ser entregues mais rapidamente em um ambiente de desenvolvimento ágil.

No próximo Capítulo serão definidos os requisitos que guiaram esta pesquisa, assim como uma comparação das linguagens apresentadas neste Capítulo em relação ao cumprimento ou não dos requisitos apresentados.

4 REQUISITOS DA LINGUAGEM

Neste Capítulo serão expostos os requisitos que guiaram o desenvolvimento desta pesquisa. Na Seção 4.1 são citadas as inspirações para que esta pesquisa começasse. Na Seção 4.2 são mostrados os requisitos primordiais deste trabalho, bem como suas justificativas. Na Seção 4.3 são descritos os requisitos finais deste trabalho. Eles foram definidos após a análise dos requisitos definidos na Seção 4.2. A comparação das linguagens quanto ao cumprimento dos requisitos apresentados neste Capítulo é feita na Seção 4.4. Finalizando este Capítulo, a conclusão é apresentada na Seção 4.5.

4.1 INSPIRAÇÕES

O trabalho realizado baseou-se em ideias exploradas do trabalho de Vilela et al. (2017). Neste trabalho os autores identificam e propõem um conjunto de características que deveriam ser suportadas em linguagens de especificação de requisitos. Outro trabalho que ajudou na concepção desta pesquisa foi o trabalho de El Ahmar et al. (2015), o qual criou uma forma de melhorar a comunicabilidade dos diagramas UML. Para tanto os autores propuseram a adição de camadas aos diagramas UML. Nestas camadas informações extras podem ser adicionadas, para que um diagrama possa comunicar mais de uma ideia para mais de um tipo de público.

O trabalho de Aslak Hellesøy (2008) criou a ideia de identificar uma forma de expandir uma linguagem que já tem sido utilizada por desenvolvedores e analistas de negócios para especificar as funcionalidades de um sistema. A extensão criada trabalha justamente na intersecção destes três trabalhos, adicionando funcionalidades a uma linguagem de especificação para que ela possa ser utilizada por mais de um público.

4.2 PRIMEIRA VERSÃO DOS REQUISITOS

A primeira versão dos requisitos para este projeto, as quais podem ser vistas na listagem a seguir, foi desenvolvida utilizando como parâmetros as inspirações discutidas na Seção 4.1 e as deficiências encontradas nas linguagens de especificação já existentes.

RL1 - Permitir a especificação de requisitos funcionais e não-funcionais

RL2 - Exportar um modelo de requisitos (Modelo de caso de uso ou História de Usuário)

RL3 - Permitir a geração de documentação (Similar ao Swagger e RAML)

RL4 - Permitir o relacionamento entre requisitos

RL5 - A linguagem deve ser de fácil compreensão para usuários não técnicos

RL6 - Permitir a comunicação entre os stakeholders

RL7 - Permitir a inclusão de dados de planejamento do projeto

RL8 - Permitir a definição de métricas para cada requisito ou grupo de requisitos

RL9 - Permitir a classificação de requisitos

RL10 - Permitir a criação de grupos de requisitos

RL11 - Permitir análise de impacto entre os requisitos e dos requisitos

RL12 - Permitir análise de longo prazo dos requisitos

RL13 - Facilitar o uso de inteligência artificial

4.2.1 Considerações sobre os requisitos

O requisito RL1, "Permitir a especificação de requisitos funcionais e não-funcionais", é essencial, pois são os requisitos funcionais que definirão o que precisa ser feito e como precisa ser feito (Helm e Wildt, 2014). Os requisitos não-funcionais são importantes para definir as restrições do sistema e prover as características do sistema para que os stakeholders fiquem satisfeitos com o sistema (Widrig e Leffingwell, 1999)(Malan et al., 2001). Desta forma, o que RL1 requisita é que a linguagem deverá ser capaz de fornecer meios para que seu usuário consiga definir os requisitos funcionais e não-funcionais de um sistema.

O requisito RL2, "Exportar um modelo de requisitos (Modelo de caso de uso ou História de Usuário)", é definido para que haja a possibilidade de exportar os requisitos para algum padrão já utilizado. Portanto, RL2 define que a nova linguagem criada deve ser modelada de forma que permita sua transformação em outro modelo.

O requisito RL3, "Permitir a geração de documentação (Similar ao Swagger e RAML)", foi inspirado nas ferramentas de documentação de API Swagger[1] e RAML[2] as quais auxiliam no processo de desenvolvimento, por gerar automaticamente a documentação do software desenvolvido.

O requisito RL4, "Permitir o relacionamento entre requisitos", foi inspirado no diagrama de requisitos do SysML, o qual é detalhado por (S. Friedenthal e Steiner, 2012), e serve para especificar a hierarquia dos requisitos. Esta hierarquia ajuda a definir o planejamento do plano de trabalho do projeto. Um relacionamento pode ser, segundo o SysML (S. Friedenthal e Steiner, 2012), de derivação, cópia, dependência, inclusão e refinamento. Portanto, o que RL4 determina é que um requisito especificado pode referenciar, utilizando os relacionamentos citados, outros requisitos.

O requisito RL5, "A linguagem deve ser de fácil compreensão para usuários não técnicos", é um pré-requisito para que o requisito RL6 seja cumprido, já que todos os stakeholders devem ser capazes de se comunicar utilizando a linguagem, ela tem que ser a mais natural possível para o usuário menos especializado em gestão de requisitos. Para tanto a linguagem precisa ser a mais parecida possível com alguma linguagem que o usuário conheça, no caso, a linguagem que mais pessoas conhecem é a linguagem natural, portanto é indicado que a linguagem precisa ser similar à linguagem natural. Como discutido na Seção 4.1, trabalhar com a comunicabilidade é o ponto central desta pesquisa, a qual teve início após a leitura do trabalho de El Ahmar et al. (2015). Por este motivo, o requisito RL5 foi definido para que a linguagem seja simples o suficiente para que todos os *stakeholders* do projeto sejam capazes de entender o requisito definido.

O requisito RL6, "Permitir a comunicação entre os stakeholders", surgiu da observação do trabalho de El Ahmar et al. (2015), já que este autor propõe formas de melhorar a comunicação de ferramentas já existentes. Outro influenciador na criação deste requisito são os métodos ágeis, os quais precisam de formas eficazes de comunicação entre seus stakeholders para que todas as principais premissas dos métodos ágeis, como adaptar-se a mudanças por exemplo, consigam ser

[1] `https://swagger.io` - Acessado em 02/04/2018

[2] `https://raml.org` - Acessado em 02/04/2018

executadas. O que o requisito RL6 determina é que a linguagem deve ser de fácil compreensão como o RL5 determina e, além disso, deve permitir a fácil negociação e edição dos requisitos.

O requisito RL7, "Permitir a inclusão de dados de planejamento do projeto", permite a definição de ordem de implementação do requisito, o tempo necessário para implementar o requisito, o número de desenvolvedores que serão alocados para desenvolver o requisito, quem são os desenvolvedores e em qual *sprint* o requisito será desenvolvido. A ordem de implementação foi inspirada pelas redes de planejamento, também conhecidas como redes de PERT-CPM (Klastorin, 2004). Já as funcionalidades de definir o número da *sprint* e o(s) desenvolvedor(es) foram inspiradas pela ferramenta Gitlab[3], que possui uma ferramenta chamada *issues*, a qual permite a definição das atividades no submenu *Board* e a definição de *sprints* no submenu *Milestones*.

O requisito RL8, "Permitir a definição de métricas para cada requisito ou grupo de requisitos", foi inspirado pelo trabalho de Costello e Liu (1995) e permite a definição de métricas para ajudar no gerenciamento de risco. Em (Costello e Liu, 1995) os autores definem algumas métricas que podem ser utilizadas, como a volatilidade dos requisitos, a completude dos requisitos, a densidade de defeitos dos requisitos, entre outras. Note que, RL8 é um requisito voltado, mas não exclusivo, para a equipe de desenvolvimento, já que definir métricas exige conhecimento do que medir e como medir.

O requisito RL9, "Permitir a classificação de requisitos", foi desenvolvido para facilitar a atribuição das tarefas entre desenvolvedores, já que será possível marcar os requisitos que compõem uma funcionalidade, possivelmente facilitando a busca dos requisito. Além disso, o trabalho de Navarro-Almanza et al. (2018) utiliza *Deep Learning* para classificar requisitos. Este trabalho pode se beneficiar de uma linguagem que dê suporte a classificação de requisitos, já que é preciso treinar algoritmos de *Deep Learning*, e caso novas bases de dados já classificadas surjam, será mais fácil treinar o algoritmo para que ele seja mais preciso. Tendo em vista o trabalho de Navarro-Almanza et al. (2018), o RL9 auxilia o cumprimento do RL13 (Facilitar o uso de inteligência artificial). O requisito RL9, portanto, é o requisito que permite seus usuários criar "etiquetas" para classificar os requisitos de sistema definidos.

O requisito RL10, "Permitir a criação de grupos de requisitos", foi definido para facilitar a definição e a referência entre múltiplos requisitos ao mesmo tempo, desta forma um grupo pode ser usado na definição de métricas ou até mesmo para servir como filtro de busca. Portanto este requisito visa melhorar a usabilidade da linguagem simplificando o uso das métricas. Da mesma forma que o RL9, o RL10 serve para marcar com "etiquetas" os requisitos de forma que eles fiquem agrupados. Por serem bastante similares a versão final dos requisitos unificou os o requisito RL9 e o RL10, como será exposto mais adiante.

Os requisitos RL11 e RL12, "Permitir análise de impacto entre os requisitos e dos requisitos" e "Permitir análise de longo prazo dos requisitos" respectivamente, foram inseridos neste projeto como uma tentativa de criar ferramentas com suporte à sustentabilidade, o conhecido como *Green IT*. A inspiração destes requisitos vieram de (Lago et al., 2015). Este tipo de análise é importante para a sustentabilidade, pois permite que seus usuários discutam sobre quais as implicações da inserção do requisito em análise no software que está sendo desenvolvido. Além disso, também é possível perceber e as implicações deste requisito no ambiente em que o software será executado. Desta forma pode-se avaliar a necessidade ou não da inserção do requisito. O requisito RL12 determina que a análise que pode ser realizada e registrada pelo requisito RL11 possa também ser feita à longo prazo. Desta forma, as consequências de implementar um determinado requisito poderá ser avaliada de forma a evitar possíveis problemas no futuro.

Por fim, o requisito RL13, "Facilitar o uso de inteligência artificial", é definido a fim de permitir que novas tecnologias possam ser integradas com a linguagem criada. Estas tecnologias

[3] `https://about.gitlab.com` - Acessado em 02/04/2018

poderão automatizar todo o processo de especificação e análise de requisitos. Já que há várias pesquisas relacionadas com processamento de linguagem natural e inteligência artificial. Como a linguagem utilizada é baseada em linguagem natural, é importante permitir que a nova linguagem possa ser estendida no futuro.

4.3 VERSÃO ATUAL DOS REQUISITOS

Após propor os treze requisitos para a linguagem, eles foram estudados e reavaliados. Nesta reavaliação alguns requisitos foram agrupados por serem similares. É o caso dos requisitos RL9 (Permitir a classificação de requisitos) e RL10 (Permitir a criação de grupos de requisitos). A escolha de agrupar os dois requisitos foi tomada pelo fato de que, pela definição do dicionário[4], uma classificação é um agrupamento, desta forma não faz sentindo existir duas seções distintas para a classificar. Desta forma, foi mantido apenas o requisito RL10, o qual se tornou o NRL9 (Permitir a criação de grupos de requisitos).

Os requisitos RL11 (Permitir análise de impacto entre os requisitos e dos requisitos) e RL12 (Permitir análise de longo prazo dos requisitos) foram agrupados em um único requisito, NRL10, pois ambos dizem respeito a permitir o usuário documentar a análise do requisito. Os demais requisitos continuam e apenas foram reescritos, criando desta forma, a listagem a seguir.

NRL1 - Permitir a especificação de requisitos funcionais e não-funcionais

NRL2 - Exportar um modelo de requisitos

NRL3 - Permitir a geração de documentação

NRL4 - Permitir o relacionamento entre requisitos

NRL5 - A linguagem deve ser de fácil compreensão para usuários não técnicos

NRL6 - Permitir a comunicação entre os stakeholders

NRL7 - Permitir a inclusão de dados de planejamento do projeto

NRL8 - Permitir a definição de métricas para cada requisito

NRL9 - Permitir a criação de grupos de requisitos

NRL10 - Permitir a documentação de análises realizadas sobre os requisitos

NRL11 - Facilitar o uso de inteligência artificial

4.4 COMPARAÇÃO DAS LINGUAGENS DA LITERATURA

Nesta Seção as linguagens apresentadas no Capítulo 3 serão comparadas quanto ao cumprimento dos requisitos finais deste trabalho apresentados na Seção 4.3 (NRLs). Para tanto a tabela da Figura 4.1 foi criada comparando as doze linguagens explicadas.

Na comparação da Figura 4.1, cada requisito (NRL) foi analisado para cada linguagem apresentada no Capítulo 3. Quando a linguagem apresentar a marca de "Funcionalidade implementada", significa que a linguagem consegue cumprir o requisito completamente. Quando

[4] `https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/classifica%C3%A7%C3%A3o/` - Acessado em 29/11/2018

Linguagem	NRL1	NRL2	NRL3	NRL4	NRL5	NRL6	NRL7	NRL8	NRL9	NRL10	NRL11	Número de ✓	Número de !	Número de ✗
Alloy	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
Alneelain	!	✓	!	✗	✗	!	✗	✗	!	✗	✓	2	4	5
Gherkin	!	✓	!	!	✓	!	✗	✗	!	✗	✓	3	5	3
i*	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
KAOS	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
Método B	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
Notação Z	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
ReSA	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
RSLingo	!	✓	!	✓	✗	!	!	✗	!	✓	✓	4	5	2
VDM	!	✓	!	!	✗	!	✗	✗	!	✗	✓	2	5	4
DASH	!	✓	!	✓	✗	!	✗	✗	!	✗	✓	3	4	4
ReqDL	✓	✓	!	✓	!	!	✗	✗	✓	✗	✓	5	3	3

Legenda:	
✓	Funcionalidade implementada
!	Funcionalidade parcialmente implementada
✗	Funcionalidade não implementada

Figura 4.1: Tabela com comparação entre linguagens

a marca de "Funcionalidade parcialmente implementada"aparece, significa que alguns pontos da funcionalidade não são cumpridos pela linguagem. Por fim, quando a marca de "Funcionalidade não implementada"aparece significa que a linguagem não suporta a o requisito analisado. É importante notar que esta análise foi realizada em 2017/2018, e que no momento da pesquisa este era o estado destas linguagens. É possível que no futuro o cumprimento dos NRLs possa mudar.

Começando a análise pelo NRL1, "Permitir a especificação de requisitos funcionais e não-funcionais", a maioria das linguagens apresentadas estão com a marca de "Funcionalidade parcialmente implementada"porque todas implementam, de alguma forma, a possibilidade de especificar requisitos funcionais, entretanto os requisitos não-funcionais não são suportados. A exceção é a linguagem ReqDL, que implementa tanto os requisitos funcionais quanto os não-funcionais.

O NRL2, "Exportar um modelo de requisitos", é cumprido por todos, já que todos eles podem ser representados de forma textual (mesmo os que utilizando elementos gráficos como i* e KAOS). Estas representações são modelos, uma vez que cada linguagem tem seus elementos bem definidos e cada uma delas é restrita a seus elementos gramaticais. Este fato permite a aplicação de técnicas de engenharia de software orientada a modelos, o que permite a transformação de cada linguagem em outra. Estas transformações entre modelos podem não ser completas, mesmo sendo possíveis. Portanto todas as linguagens exportam modelos de requisitos.

O NRL3, "Permitir a geração de documentação", é parcialmente cumprido porque, é possível considerar que a própria especificação é parte da documentação. Então mesmo que as linguagens não tenham como característica a geração de documentação, é possível considerar este NRL como parcialmente cumprido.

O NRL 4, "Permitir o relacionamento entre requisitos", é cumprido pela maioria das linguagens, as exceções são: Alneelain, Gherkin e VDM. No caso do Alneelain, nenhuma estrutura pode ser utilizada para realizar relacionamentos, entretanto, nos casos do Gherkin e VDM é possível abusar das notações e definir parcialmente as relações entre os requisitos. Este abuso de notação, entretanto, não é recomendado, pois pode prejudicar outros elementos da linguagem.

O NRL 5, "A linguagem deve ser de fácil compreensão para usuários não técnicos", somente foi cumprido pelo Gherkin, já que o Gherkin utiliza a linguagem natural para especificar requisitos. Todas as outras linguagens citadas utilizam uma sintaxe própria e diferente da linguagem natural. Mesmo quando a linguagem utiliza linguagem natural, como a ReqDL, ela tem detalhes gramaticais que podem dificultar um pouco o entendimento, como no caso da ReqDL, a estrutura envolto de chaves ({}) e a topicalização do itens do requisito podem não ser intuitivos para o usuário não técnico.

O NRL6, "Permitir a comunicação entre os stakeholders", todos os requisitos cumprem parcialmente, porque mesmo que a linguagem seja formal e que um usuário não técnico não consiga entender, os desenvolvedores do sistema conseguiram entender e como os desenvolvedores também são stakeholders, logo há comunicação entre stakeholders. Entretanto, o que estas linguagens em geral não fornecem é a comunicação entre diferente tipos de desenvolvedores. Este é o motivo de estas linguagens receberem a marcação de "Funcionalidade parcialmente implementada".

O NRL 7, "Permitir a inclusão de dados de planejamento do projeto", não é implementada por nenhuma linguagem, pois todas elas tem um foco específico (rastreabilidade, completude, executabilidade, etc.) fazendo com que a gerência de projeto seja controlada por outra ferramenta. Similarmente, o NRL 8,"Permitir a definição de métricas para cada requisito", não é cumprido por nenhuma linguagem encontrada na literatura.

O NRL 9, "Permitir a criação de grupos de requisitos", tem como maioria das linguagens marcado como "Funcionalidade parcialmente implementada", porque mesmo que a linguagem não especifique uma forma de agrupar os requisitos, geralmente é possível dividir os requisitos e diretórios, ou ainda, nomear cada requisito de forma que uma espécie de agrupamento seja criada. Como exposto na explicação da comparação do NRL 4, as técnicas acima descritas podem ser um abuso de notação, e podem prejudicar outros elementos da linguagem.

O NRL 10, "Permitir a documentação de análises realizadas sobre os requisitos", é implementado apenas pelo RSLingo. Todas as outras linguagens não suportam esta funcionalidade, pois elas apenas documentam o requisito e não artefatos acessórios aos requisitos.

O NRL 11, "Facilitar o uso de inteligência artificial", é implementado por todas as linguagens, pois, como discutido anteriormente, por se tratar de modelos, é possível transformar qualquer modelo em outro, inclusive em um modelo que seja facilmente processado por algoritmos de inteligência artificial.

4.5 CONCLUSÃO

Neste Capítulo foram apresentadas, na Seção 4.1, as inspirações que fizeram este pesquisa começar. Após as inspirações, a primeira versão dos requisitos deste trabalho foi explicada na Seção 4.2. A evolução destes requisitos foi apresentada na Seção 4.3. Finalizando o Capítulo, uma comparação foi realizada com as linguagens apresentadas no Capítulo 3 e os requisitos do trabalho apresentados na Seção 4.3.

No Capítulo 5 será especificada a arquitetura da solução proposta pelo autor para cumprir todos os requisitos apresentados, já que nenhuma das linguagens encontradas na literatura cumprem integralmente os requisitos.

5 ARQUITETURA

Neste Capítulo serão expostas todas as partes que compõem o trabalho, assim como a as mudanças realizadas na linguagem criada e apresentada na qualificação em relação a versão atual. Começando na Seção 5.1 com a definição dos itens da linguagem, seguindo com o interpretador da extensão na Seção 5.2.

5.1 DEFINIÇÃO DA LINGUAGEM

O Gherkin Specification Extension (GSE) é composto de sete seções. Cada seção foi definida a fim de cumprir os requisitos especificados na Seção 4. As seções definidas são:

1. *Description* (Descrição)
2. *Group* (Grupos)
3. *Contraints and qualities* (Restrições e qualidades)
4. *Relationship* (Relacionamento)
5. *Planning* (Planejamento)
6. *Metrics* (Métricas)
7. *Notes* (Notas)

Conforme exposto na Seção 4, foram definidos alguns requisitos que a linguagem e a ferramenta criada cumprissem. Estes requisitos, assim como o que cumprirá cada seção é mostrado na tabela da Figura 5.1.

Quando a tabela da Figura 5.1 diz que o requisito é cumprido por "Gherkin e GSE" (requisitos 1, 3 e 7) significa que a linguagem como um todo implementa o requisito citado. Por exemplo, o requisito (1) que diz que a linguagem deve permitir a comunicação entre os stakeholders, o GSE cumpre este requisito com o uso das seções enumeradas acima (que serão detalhadas a seguir) e o Gherkin cumpre este requisito especificando os cenários de cada requisito.

Quando um requisito é cumprido pelo "Interpretador"significa que quem implementa o requisito não é a linguagem em si, mas sim o software criado para interpretar a linguagem GSE. Isto acontece no requisito (3) que diz que a linguagem deve permitir a geração de documentação. O outro caso que o interpretador aparece como responsável por cumprir um requisito, é no requisito (11) que afirma que a linguagem deve facilitar o uso de inteligência artificial. Neste caso, o interpretador é responsável por gerar arquivos que são mais fáceis de processar (`.csv` e `.json`) utilizando algoritmos de inteligência artificial.

De uma forma geral, um arquivo que utiliza o GSE deve seguir a gramática definida no Código 5.1. A gramática criada usa a notação da ferramenta textX[1] para criar o meta-modelo da gramática GSE.

[1] `http://www.igordejanovic.net/textX/stable/grammar/` - Acessado em 06/11/2018

Identificação do Requisito	Requisitos	Cumprido por
NRL1	Permitir a especificação de requisitos funcionais e não-funcionais	Seções Description/Rules do GSE
NRL2	Exportar um modelo de requisitos	Gherkin e GSE
NRL3	Permitir a geração de documentação	Interpretador do GSE
NRL4	Permitir o relacionamento entre requisitos	Seção Relationship do GSE
NRL5	A linguagem deve ser de fácil compreensão para usuários não técnicos	Gherkin e GSE
NRL6	Permitir a comunicação entre os stakeholders	Gherkin e GSE
NRL7	Permitir a inclusão de dados de planejamento do projeto	Seção Planning do GSE
NRL8	Permitir a definição de métricas para cada requisito	Seção Metrics do GSE
NRL9	Permitir a criação de grupos de requisitos	Seção Group do GSE
NRL10	Permitir a documentação de análises realizadas sobre os requisitos	Seção Notes do GSE
NRL11	Facilitar o uso de inteligência artificial	Interpretador do GSE e GSE

Figura 5.1: Requisitos da pesquisa e como serão cumpridos

```
GSE_MODEL:
   Description=DESCRIPTION?
   (
      Group=GROUP?
      Constraints=CONSTRAINTS?
      Relationship=RELATIONSHIP?
      Planning=PLANNING?
      Metrics=METRICS?
   )#
   Notes=NOTES?
;
```

Código 5.1: Estrutura da gramática do GSE

5.1.1 Description

Na seção *Description* é possível especificar o ator do requisito, uma funcionalidade que deverá ser implementada e a motivação para implementar esta funcionalidade. Para tanto é utilizado o padrão Connextra de Histórias de Usuário, desta forma um requisito pode ser utilizando a estrutura definida pela gramática do Código 5.2.

```
DESCRIPTION:
   'Description:' NEWLINE
   'As' Stakeholder=TEXT NEWLINE
```

```
4     ('Or' Stakeholder+=TEXT NEWLINE)*
5     'I want to' Action=TEXT NEWLINE
6     'So that I' Motivation=TEXT NEWLINE
7  ;
```

Código 5.2: Estrutura da gramática da seção Description

Utilizando a gramática do Código 5.2 pode ser escrita um descrição como mostra o Código 5.3

```
1  (*\bfseries Description:*)
2     (*\bfseries As*) an actor
3     (*\bfseries I want to*) do something
4     (*\bfseries So that I*) can do my stuff better
```

Código 5.3: Exemplo de uso da seção Description

5.1.2 Group

A seção *Group* permite ao usuário agrupar informações de forma a ter uma maior organização dos requisitos. Além disso, o relatório que pode ser gerado pelo interpretador utiliza a seção *Group* para contabilizar quantos requisitos existem em cada grupo. Note que nesta contagem um requisito pode ser contado mais de uma vez caso ele possua mais de um grupo. A seguir é exposta a gramática da seção *Group* no Código 5.4.

```
1  GROUP:
2     'Group:' NEWLINE
3     Groups+=GROUP_TAG+ NEWLINE
4  ;
```

Código 5.4: Estrutura da gramática da seção Group

Um exemplo de como utilizar a seção *Group* é exibido no Código 5.5.

```
1  (*\bfseries Group:*)
2     Group A, Group B, Group C
```

Código 5.5: Exemplo de uso da seção Group

5.1.3 Contraints and qualities

A seção *Contraints and qualities* é a responsável por especificar os requisitos não funcionais da funcionalidade especificada. Isto significa que é preciso especificar tudo o que o requisito deve implementar e tudo que o ator alvo (especificado na seção *Description*) não deve ser capaz de fazer. Para tanto a estrutura mostrada pela gramática do Código 5.6 deve ser utilizada:

```
1  CONSTRAINTS:
2     'Constraints and Qualities:' NEWLINE
3     ('The requirement should implement:' NEWLINE Qualities+=ITEM+ NEWLINE)?
4     ('The actor should not be able to:' NEWLINE Constraints+=ITEM+ NEWLINE)?
5  ;
```

Código 5.6: Estrutura da gramática da seção Constraints and Qualities

O exemplo da utilização desta gramática é mostrado no Código 5.7:

```
(*\bfseries Constraints and Qualities:*)
   (*\bfseries The requirement should implement:*)
   (*\bfseries **) Item 1 - A
   (*\bfseries The actor should not be able to:*)
   (*\bfseries **) Item 1 - B
      (*\bfseries >*) Subitem 1.1 - B
   (*\bfseries **) Item 2 - B
```

Código 5.7: Exemplo de uso da seção Constraints and Qualities

Caso um requisito não-funcional não esteja relacionado apenas a um requisito funcional, e sim com um grupo de requisitos, este requisito não-funcional deverá ser especificado em todos os arquivos os quais ele deverá ser implementado. Uma solução mais elegante será desenvolvida em trabalho futuro.

5.1.4 Relationship

Na seção *Relationship* da linguagem, é possível definir como a funcionalidade se relaciona com outras funcionalidades. As relações possíveis são:

- Derivação
- Contido em
- Cópia
- Refinamento
- Dependência

A relação de derivação implica que a funcionalidade a qual o usuário está lendo é derivada das funcionalidades listadas no item de derivação. Da mesma forma, os demais itens seguem a mesma lógica.

Estas cinco classes de relacionamentos são também existem no diagrama de requisitos do SysML (S. Friedenthal e Steiner, 2012). No GSE, para definir relacionamentos usa-se a gramática definida no Código 5.8

```
RELATIONSHIP:
   'Relationship:' NEWLINE
   'This requirement is:' NEWLINE
   (
      ('* Derived from requirement(s)' SPACE
         Derived=REQUIREMENT_LIST NEWLINE)?
      ('* Contained in requirement(s)' SPACE
         Contains=REQUIREMENT_LIST NEWLINE)?
      ('* A copy of the requirement(s)' SPACE
         Copy=REQUIREMENT_LIST NEWLINE)?
      ('* A refinament of the requirement(s)' SPACE
         Refinament=REQUIREMENT_LIST NEWLINE)?
      ('* Dependent on requirement(s)' SPACE
         Dependent=REQUIREMENT_LIST NEWLINE)?
   )#
;
```

Código 5.8: Estrutura da gramática da seção Relationship

Utilizando a gramática do Código 5.8 obtemos o exemplo do Código 5.9

```
(*\bfseries Relationship:*)
  (*\bfseries This requirement is:*)
   (*\bfseries * Derived from requirement(s)*) req 1, req2, req 1 2, req 3
   (*\bfseries * Contained in requirement(s)*) req a, req2, req 1 2, req 3
   (*\bfseries * A copy of the requirement(s)*) req c, req2, req 1 2, req 3
   (*\bfseries * A refinament of the requirement(s)*) req f, req2, req 1 2, req 3
   (*\bfseries * Dependent on requirement(s)*) req d, req2, req 1 2, req 3
```

Código 5.9: Exemplo de uso da seção Descrição

5.1.5 Planning

A seção *Planning* é utilizada para definir dados referente ao planejamento de projeto. Os dados que são informados nesta seção dizem respeito à ordem de implementação da funcionalidade, ao tempo necessário para desenvolvê-la, ao número de desenvolvedores que deverão ser alocados para implementá-la, ao nome dos desenvolvedores responsáveis pela implementação e por fim à sprint[2] a qual é esperada que a funcionalidade seja desenvolvida. Todas estas informações são escritas em um texto padrão o qual pode ser visualizado na gramática do Código 5.10:

```
PLANNING:
   'Planning:' NEWLINE
   'This requirement will be implemented' SPACE Order=ORDINAL_NUM SPACE
   'and will take' SPACE Time_needed=TIME SPACE
   'to be implemented by' SPACE Num_of_developers=/\d+/
   SPACE 'developer(s).' NEWLINE
   'The developer(s) responsible for implementing this requirement is/are'
   SPACE Developers+=WORD_WITH_SPACE[','] SPACE '.' NEWLINE
   'The sprint the will implement this feature is:'
   SPACE Sprint=WORD_WITH_SPACE SPACE '.' NEWLINE
;
```

Código 5.10: Estrutura da gramática da seção Planning

O resultado do uso da gramática do Código 5.10 é o exemplo mostrado no Código 5.11.

```
(*\bfseries Planning:*)
  This requirement will be implemented (*\bfseries 8th*) and will take (*\bfseries 60
  implemented by (*\bfseries 1*) developer(s). The developer(s) responsible for
  implementing this requirement is/are (*\bfseries Mariah Cash, Josh Puddle*).
  The sprint that will implement this feature is: (*\bfseries Final Sprint*).
```

Código 5.11: Exemplo de uso da seção Planning

5.1.6 Metrics

A seção *Metrics* é a seção responsável por definir quais métricas deverão ser usadas para avaliar a funcionalidade e, caso o redator da funcionalidade queira, qual é o valor esperado da métrica. A estrutura desta seção é definida pela gramática do Código 5.12

```
METRICS:
   'Metrics:' NEWLINE
```

[2]Uma sprint é uma fase do projeto na metodoligia ágil SCRUM. Desta forma em uma sprint é definido um conjunto de atividades que deverão ser realizadas em determinado período de tempo - https://gaea.com.br/o-que-e-sprint-entenda-como-agilizar-as-entregas-de-ti/ - Acessado em 07/12/2018

```
3     'The metrics used to evaluate this requirement are:' NEWLINE
4     (Metric_item+=METRIC_ITEM)+ NEWLINE
5 ;
```

Código 5.12: Estrutura da gramática da seção Metrics

Para exemplificar o uso da seção *Metrics* foi escrito o Código 5.13

```
1 (*\bfseries Metrics:*)
2   (*\bfseries The metrics used to evaluate this requirement are:*)
3    (*\bfseries **) NOH;
4    (*\bfseries **) CMPLX(*\bfseries ; The expected value is at least*) 50 lines
```

Código 5.13: Exemplo de uso da seção Metrics

5.1.7 Notes

A seção *Notes* é o lugar que permite o usuário escrever ao que ele quiser sobre a funcionalidade, inclusive uma análise do requisito especificado, como requisitado pela Figura 5.1. Esta seção não possui nenhuma estrutura fixa, a não ser seu cabeçalho. Esta seção pode ser usada para documentar, além das análises citadas, dificuldades encontradas durante a implementação, detalhes do negócio do cliente ou qualquer outra informação pertinente ao requisito. Sua gramática é definida pelo Código 5.14

```
1 NOTES:
2    'Notes:' NEWLINE
3    (Note+=TEXT NEWLINE)+
4 ;
```

Código 5.14: Estrutura da gramática da seção Notes

Uma possível forma de utilizar esta seção é mostrada no Código 5.15

```
1 (*\bfseries Notes:*)
2   You can type whatever you want here.
```

Código 5.15: Exemplo de uso da seção Notes

5.2 INTERPRETADOR DA EXTENSÃO

O interpretador da linguagem[3] acima foi criado usando a meta-linguagem para especificação de linguagem específica de domínio (DSL) textX[4]. O textX é distribuído como uma biblioteca de Python. Uma de suas funcionalidades é a capacidade de exportar o meta-modelo criado para formato *.dot*. Este arquivo exportado pode ser importado pela ferramenta *Graphviz*[5] para gerar uma imagem do meta-modelo. O meta-modelo exportado da linguagem criada é mostrado na Figura 5.2

A gramática criada pode ser encontrada no Apêndice I. Com este meta-modelo é possível coletar as informações importantes de uma instância da linguagem (a especificação propriamente dita) para gerar relatórios e exportar os dados dos requisitos para que outros softwares possam utilizar.

[3]Disponível em `https://github.com/HenriqueVarellaEhrenfried/GSE-Interpreter` - Acessado em 09/03/2019

[4]A documentação para o textX pode ser encontrada em `http://www.igordejanovic.net/textX/stable/` - Acessado 09/10/2018

[5]Disponível em `https://www.graphviz.org/` - Acessado em 09/03/2019

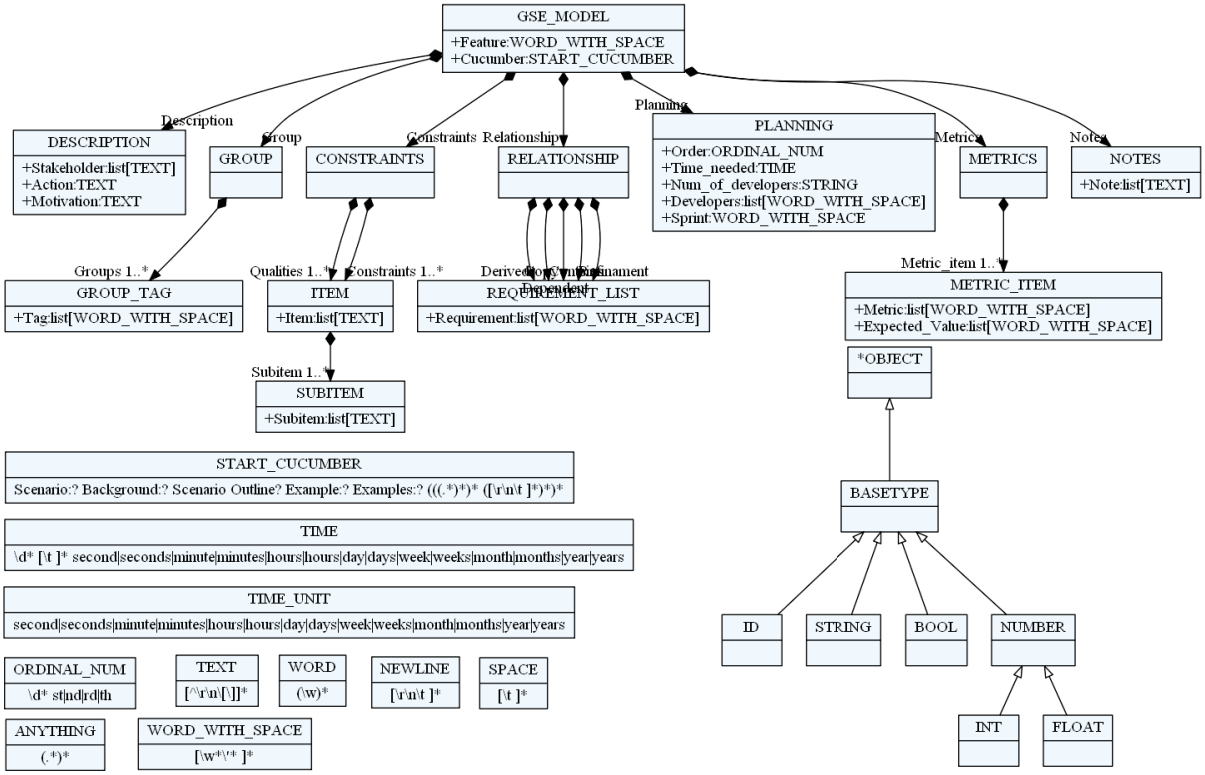


Figura 5.2: Meta-modelo da linguagem GSE

As funcionalidades implementadas no interpretador são:

1. Criação de arquivo CSV contendo todos os dados dos arquivos selecionados
2. Criação de arquivo JSON contendo todos os dados dos arquivos selecionados
3. Criação de relatório em PDF

As duas primeiras funcionalidades, criação de arquivos CSV e JSON, têm como objetivo exportar os requisitos para que outros softwares possam utilizar os requisitos especificados. Para utilizar estas funcionalidades, basta invocar o interpretador passando como parâmetro a funcionalidade desejada (criar CSV ou JSON), uma lista de arquivos ou um diretório que contém os arquivos e o arquivo de saída. Há ainda parâmetros opcionais caso esteja criando arquivo CSV. Estes parâmetros são o separador entre colunas e a codificação de saída do arquivo.

A última funcionalidade, a criação de relatório em PDF, permite ao utilizador criar um relatório que contém, além dos requisitos especificados, os itens a seguir:

- A descrição do projeto
- O número de requisitos
- O número de atores diferentes
- A lista de atores
- O número de ocorrência de cada ator
- O número de desenvolvedores

- A lista de desenvolvedores
- O número de vezes requisitos que cada desenvolvedor irá desenvolver
- O tempo estimado para implementar o projeto
- Quantas *sprints* o projeto terá
- O número de requisitos por grupo
- Um grafo de dependência dos requisitos
- A lista de métricas que serão utilizadas no projeto
- Os requisitos

Este relatório também pode ser acessado como uma página HTML, já que para gerar o relatório em PDF é preciso gerar um modelo em HTML para que uma biblioteca transforme este HTML em PDF. Portanto, se o usuário quiser ele poderá utilizar o relatório no formato HTML.

Para gerar o relatório, é preciso que o usuário preencha dois arquivos de configuração para o projeto. O primeiro, permite ao usuário inserir o nome da instituição, o logotipo logo da instituição, o nome do projeto, o logotipo do projeto, a versão do projeto e a descrição do projeto. O segundo arquivo permite ao usuário preencher os dados referente às métricas. Neste caso, para cada métrica o usuário pode escrever o acrônimo da métrica, seu nome, como ela funciona e a forma que ela deve ser interpretada.

Para a funcionalidade de gerar relatório em PDF, basta invocar o interpretador passando como parâmetro a funcionalidade de geração de relatório, uma lista de arquivos ou um diretório que contém os arquivos, e o arquivo de saída.

O funcionamento básico do interpretador, independentemente da funcionalidade, consiste em ler o arquivo da especificação, extrair as informações utilizando o meta-modelo criado, e então invocar a funcionalidade requisitada pelo usuário.

A inteligência artificial é suportada, além da modelagem da linguagem, pela geração de arquivos CSV e JSON pelo interpretador da linguagem. Estes tipos de arquivos já possuem bibliotecas que permitem sua manipulação de forma mais fácil em diversas linguagens de programação como Python, Ruby, Java e outras. Desta forma o uso dos dados escritos utilizando o GSE são obtidos mais facilmente para o treinamento de algoritmos de aprendizagem de máquina, facilitando, por exemplo o treinamento de um algoritmo de classificação de História de Usuários.

5.3 CONCLUSÃO

Neste Capítulo, a linguagem GSE foi definida na Seção 5.1, a arquitetura do interpretador da extensão foi abordada na Seção 5.2. Desta forma, é possível apresentar o estudo de caso realizado para validar se o GSE é de fácil utilização por pessoas não técnicas. Este estudo de caso é detalhado no Capítulo 6.

6 ESTUDO DE CASOS

Nesta Seção serão conduzidos um estudo de casos múltiplo para validar se o GSE atende o requisito de ser de fácil utilização por usuários não técnicos. Para tanto, a Seção 6.1 apresentará os como serão realizados os estudos de casos e seus participantes. A Seção 6.2 determinará os objetivos dos estudos de casos, bem como os resultados esperados. A Seção 6.3 detalhará os resultados obtidos nos estudos de casos e os discutirá. As ameaças à validade dos estudos de casos são expostas na Seção 6.4. As percepções do autor durante os estudos de casos são descritas na Seção 6.5. Por fim, a Seção 6.6 conclui o Capítulo 6 expondo as conclusões obtidas com os estudos de casos.

6.1 PLANEJAMENTO DOS ESTUDOS DE CASOS

Esta Seção explica como foi realizado o planejamento dos estudos de casos. Desta forma, esta Seção aborda como foram escolhidas as turmas que participaram dos estudos de casos, a instrumentação utilizada, as etapas dos estudos de casos, os objetivos da pesquisa, os resultados obtidos e as ameaças a validade dos estudos de casos.

6.1.1 Participantes

Os estudos de casos foram realizados com duas turmas diferentes, a primeira turma continha dez pessoas do curso de Tecnologia em Comunicação Institucional da Universidade Federal do Paraná. Sendo que dois participantes são professores do curso e os outros oito participantes são estudantes. Os estudantes e professores do curso de comunicação se caracterizam por estudar formas de implantar políticas de comunicação, para tanto eles precisam saber planejar e organizar eventos, traduzir textos técnicos, criar e editar publicações internas e externas, assessorar negociações nacionais e internacionais e revisar textos.

Os voluntários participam da Agência Ziip[1], portanto há estudantes de diversos períodos, sendo que boa parte dos estudantes já cursaram mais da metade do curso. Devido às características do curso, há contato com softwares e sistemas de informação desde o primeiro período, contudo eles não são treinados em especificar sistemas, o que pode caracterizá-los como possíveis clientes em um projeto de software.

Os integrantes da turma de tecnologia em comunicação institucional foram sugeridos pela coordenadora do curso em reunião anterior a execução do estudo de caso. Nesta reunião foram explicados os requisitos da pesquisa: Pessoas que utilizam computadores para trabalho em aplicações mais complexas que editores de texto e criadores de apresentações.

A outra turma continha estudantes dos cursos de Ciência da Computação e Informática Biomédica da Universidade Federal do Paraná. Estes estudantes participam do grupo Programa de Educação Tutorial (PET) de informática[2] e foi composto por dez voluntários.

Os participantes do PET Informática que participaram estão nos primeiros períodos de seus cursos e não haviam estudado engenharia de requisitos, teoria de sistemas, tampouco nenhuma disciplina relacionada com engenharia de software.

[1]Disponível em `http://agenciaziip.wixsite.com/ziip` - Acessado em 26/10/2018
[2]`http://pet.inf.ufpr.br/doku.php?id=start` - Acessado em 30/10/2018

A turma do PET Informática foi questionada antes do estudo de caso quanto ao conhecimento em sistemas e em especificação de software. Todos os voluntários afirmaram não ter conhecimento nesta área, e portanto foram classificados aptos a participar da pesquisa.

6.1.2 Instrumentação

Neste estudos de casos foram utilizados quatro artefatos. Um dos artefatos utilizados foi uma apresentação de slides para realizar os estudos de casos, a qual pode ser visualizada no Apêndice I. Outro artefato utilizado foi o Termo de Consentimento Livre e Esclarecido (TCLE). Este termo de consentimento também está disponível no Apêndice I.

Foi utilizado, também, uma especificação de software como problema para que os participantes pudessem utilizar a linguagem. O sistema foi definido em conjunto com a coordenadora do curso de tecnologia em comunicação institucional e formalizado pelo autor. A definição do sistema pode ser visto na Figura 6.1. Esta parceria para decidir qual sistema seria utilizado na pesquisa foi realizado para que o software que seria especificado utilizando o GSE fizesse sentido para os participantes da pesquisa.

```
O calendário setorial é um software que permite um setor de uma instituição
agendar eventos, reuniões e palestras.  Neste sistema, qualquer professor ou
técnico administrativo pode incluir um evento.

Este calendário possui um moderador que é capaz de adicionar usuários, remover
usuários e remover eventos. Um usuário padrão pode adicionar um evento e remover
os eventos que ele mesmo criou. Os usuários são cadastrados pelo moderador. Cada
cadastro contém o nome, o curso, o e-mail e a senha do usuário.

Quando um evento é agendado, o usuário que incluiu este agendamento foi
requisitado a inserir o nome do evento, seu tipo, horário e local.

A exibição dos eventos agendados poderá ser feita por mês, por semana ou por curso
por qualquer pessoa que acesse o calendário setorial.
```

Figura 6.1: Especificação do Calendário Setorial

Para a realização da avaliação da linguagem, foi aplicado um questionário baseado na terceira versão do modelo de aceitação de tecnologia (TAM 3 [sigla em inglês para Technology Acceptance Model]), detalhado por Venkatesh e Bala (Venkatesh e Bala, 2008). O TAM é uma técnica que permite avaliar o nível de adoção de uma tecnologia por seus usuários. Para tanto o TAM trabalha com a percepção de utilidade e percepção de facilidade de uso. As perguntas utilizadas para avaliar estes aspectos do GSE, contendo inicialmente o identificador da pergunta (FUP, UP, IP), são:

- **Facilidade de uso percebida**

FUP1 - Minha interação com a linguagem GSE foi clara e compreensível

FUP2 - Interagir com a linguagem GSE não exige muito do meu esforço mental

FUP3 - Considero a linguagem GSE fácil de usar

FUP4 - Considero fácil utilizar a linguagem GSE para fazer o que eu quero que ela faça, especificar sistemas de forma que todos os stakeholders entendam a especificação

- **Utilidade percebida**

UP1 - Usar a linguagem GSE melhorou o meu desempenho na especificação de requisitos de sistemas

UP2 - Usar a linguagem GSE permitiu aumentar minha produtividade na especificação de requisitos de sistemas

UP3 - Usar a linguagem GSE aumentou minha eficácia na especificação de requisitos de sistemas

UP4 - Considero a linguagem GSE útil para a especificação de requisitos de sistemas

- **Intenção de uso**

IP1 - Supondo que eu tenho acesso à linguagem GSE, eu pretendo usá-la

IP2 - Levando em conta que eu tenho acesso à linguagem GSE, eu prevejo que eu iria usá-la

As questões relativas a facilidade de uso percebida fornecem o nível de facilidade que o usuário experienciou ao utilizar a linguagem. As perguntas sobre utilidade percebida exprime o quão útil o usuário achou a linguagem. Por fim as questões com a temática de intenção de uso mensuram o quanto os usuários estariam dispostos em utilizar novamente a linguagem apresentada caso tivessem a chance de utilizá-la.

Como o sucesso da linguagem depende da sua aceitação, e para que a linguagem seja aceita ela precisa ser fácil de usar e deve agregar o valor de utilidade em seus usuários, o TAM é o modelo adequado para avaliar a linguagem, pois além de avaliar as características necessárias, avalia também a intenção de uso futuro da linguagem. Este último também é decorrente da facilidade de uso percebida e da utilidade percebida.

Caso as três características sejam avaliadas positivamente, o objetivo da linguagem foi alcançado, foi criada uma linguagem que o usuário acha fácil, útil e que portanto, tem intenção de utilizá-la novamente. Isso significará que os usuários, na posição de *stakeholders* de projetos de software, adotariam a tecnologia segundo o modelo do TAM.

6.1.3 Etapas dos estudos de casos

Os estudos de casos foram conduzidos em duas datas distintas. A turma de comunicação institucional participou do estudo de caso no dia 10 de Outubro de 2018, enquanto que a turma do PET Informática participou do estudo de caso no dia 29 de Outubro de 2018. Entretanto os estudos de casos foram conduzido da mesma forma e utilizou a mesma instrumentação nas duas datas.

O estudo de caso foi composto de três etapas, a primeira etapa foi a de treinamento dos participantes, detalhada na Seção 6.1.3.1 - **Treinamento**. Após o treinamento, foi proposto um exercício, detalhado na Seção 6.1.3.2 - **Exercício proposto**. Por fim o participantes responderam um questionário, como expõem a Seção 6.1.3.3 - **Questionário pós-uso**.

6.1.3.1 Treinamento

O treinamento, cuja duração foi de aproximadamente 35 minutos, foi realizado em português assim como a especificação do sistema porque foi preciso que todos os voluntários pudessem entender a linguagem, e como o autor não sabia de antemão o nível do inglês dos voluntários, a linguagem foi traduzida de forma a permitir a compreensão de todos. Nesta

tradução não houve comprometimento algum do ponto de vista de utilização da linguagem em relação à versão original em inglês.

O único ponto em que houve comprometimento foi na utilização do interpretador, o qual não pode ser usado com as especificações escritas pelos voluntários, pois o interpretador foi inicialmente desenvolvido para funcionar com a gramática em inglês. Há planos de futuramente aumentar os idiomas suportados pelo GSE, assim como acontece com o Gherkin, o qual suporta 74 idiomas diferentes[3]. Entretanto este comprometimento não afeta os resultados da avaliação da linguagem em si, pois ela ainda pode ser utilizada. Apenas os artefatos que o interpretador geraria não poderão ser gerados. Estes artefatos são documentais e extraídos das definições criadas utilizam a linguagem.

O Treinamento foi composto de duas partes. A primeira parte apresentou os conceitos básicos necessários para que os participantes pudessem entender os conceitos que o GSE utiliza. Estes conceitos são: Sistemas e Requisitos. A segunda etapa foi composta da apresentação do GSE e explicação de como utilizar o GSE.

Durante a segunda etapa, foram apresentados apenas as seções do GSE que o usuário não técnico utilizaria. Isto compreende as seções *Feature* (Traduzido como Característica), *Description* (Traduzido como Descrição), *Constraints and Qualities* (Traduzido como Restrições e Qualidades), *Notes* (Traduzido como Notas) e *Scenarios* (Traduzido como Cenários). A parte de cenários é implementada pelo *Gherkin* e, portanto, não foi desenvolvida pelo GSE, uma vez que já está desenvolvida.

6.1.3.2 Exercício proposto

O exercício proposto ao final do treinamento foi a especificação do sistema especificado na Figura 6.1 utilizando o GSE. Assim como aconteceu no treinamento, tanto o sistema requisitado quanto a solução fornecida pelos participantes foram escritos em português. O tempo de execução desta etapa foi de aproximadamente 50 minutos.

Para realizar o exercício proposto, foi requisitado que os voluntários se reunissem em grupos de até quatro pessoas a fim de terem com quem discutir suas soluções. A forma de formação de grupos foi através da proximidade entre os locais que os participantes estavam acomodados durante o treinamento. Então, para cada grupo foi pedido que especificassem o sistema proposto (sistema da Figura 6.1), de forma que cada integrante do grupo especificasse ao menos um requisito utilizando os componentes aprendidos do GSE. O requisito que cada integrante do grupo especificou foi definido pelo grupo que ele fez parte, de forma que o autor não interferiu na autonomia do grupo.

A qualquer momento que fosse necessário, os participantes podiam requisitar ajuda do autor para tirar dúvidas relacionadas a linguagem. Em momento algum o autor interferiu na solução dos participantes, desta forma perguntas do estilo *"Isto aqui está certo?"* não foram respondidas, entretanto perguntas do estilo *"Como funciona a seção de Descrição?"* foram respondidas quantas vezes fossem necessárias.

6.1.3.3 Questionário pós-uso

O questionário pós-uso foi baseado no TAM de versão 3, como apresentado na Seção 6.1.2. Assim que os participantes acabavam as especificações que estavam criando eles recebiam o questionário pós-uso.

[3] `https://docs.cucumber.io/gherkin/reference/#spoken-languages` - Acessado em 21 de Novembro de 2018

Este questionário foi respondido de forma individual, sem interferência de ninguém, desta forma ele representa as opiniões individuais dos participantes. O tempo aproximado de duração desta etapa do estudo de caso foi de 15 minutos.

Após responder o questionário, os participantes retornavam para o autor tanto a especificação que criaram quanto o questionário respondido. Com a entrega destes artefatos, o estudo de caso era concluído com o participante.

Cada pergunta podia ser respondida utilizando uma escala baseada na escala de Likert (Likert, 1932) composta de sete possibilidades: Concordo Totalmente, Concordo Amplamente, Concordo Parcialmente Não concordo nem discordo, Discordo Parcialmente, Discordo Amplamente e Discordo Totalmente. Estes sete níveis auxiliam os participantes a definir mais precisamente seu grau de concordância com a afirmação fornecida.

6.2 OBJETIVO DOS ESTUDOS DE CASOS

O objetivo destes estudos de casos é verificar se é possível criar uma linguagem de fácil utilização por pessoas não técnicas que possa ser computável, de forma que a comunicação entre usuários e desenvolvedores aconteça com menos ruído do que as linguagens naturais.

6.2.1 Resultados esperados

Como foi utilizado o TAM 3 para avaliar a aceitação do GSE pelos usuários, será medida a facilidade de uso percebida, a utilidade percebida e a intenção de uso.

O esperado é que a avaliação dos usuários no quesito facilidade de uso os usuários forneça opiniões mistas. Isso porque alguns voluntários têm mais facilidade em aprender do que outros, esses podem precisar mais tempo em contato com a linguagem para conseguir entendê-la.

Quanto às perguntas sobre utilidade percebida, é esperado que os usuários reportem que encontraram utilidade na linguagem, pois como são estudos de casos controlados, os usuários perceberão logo que começarem a entender a linguagem que com ela é possível especificar coisas de uma forma clara. Além disso, pelo perfil dos voluntários escolhidos, é pouco provável que eles conheçam outras formas de especificar um sistema, e geralmente as pessoas tendem a achar conhecimentos novos úteis.

Por fim, nas perguntas de intenção de uso, é esperado que os usuários afirmem querer utilizar a linguagem, pelos mesmos motivos citados em utilidade percebida. Portanto, é esperado que, de uma maneira geral, a linguagem seja bem avaliada.

6.3 RESULTADOS OBTIDOS

Nesta Seção serão apresentados os resultados dos estudos de casos executados. Os resultados da turma de comunicação institucional serão apresentados na Seção 6.3.1 e os resultados da turma do PET Informática serão mostrados na Seção 6.3.2. Em sequência, será apresentada uma discussão dos resultados na Seção 6.3.3.

A atribuição de valores para cada opção, seguindo a tabela da Figura 6.4 permite analisar a avaliação da linguagem de forma quantitativa, já que será possível calcular, por exemplo, uma nota média da linguagem, seu desvio padrão e a distribuição normal das seções do questionário. Estes cálculos estatísticos fornecem mais material para que uma avaliação precisa do desempenho da linguagem seja realizada.

6.3.1 Turma de comunicação institucional

Iniciando a análise de respostas por pergunta, a tabela da Figura 6.2 mostra quantas respostas de cada tipo cada questão obteve.

ID da Pergunta	Número de Concordo Totalmente	Número de Concordo Amplamente	Número de Concordo Parcialmente	Número de Não Concordo Nem Discordo	Número de Discordo Parcialmente	Número de Discordo Amplamente	Número de Discordo Totalmente	Número de Respostas
FUP1	0	3	6	0	1	0	0	10
FUP2	0	2	3	0	3	1	1	10
FUP3	0	5	3	0	1	1	0	10
FUP4	4	1	3	0	1	1	0	10
UP1	1	6	2	0	1	0	0	10
UP2	1	4	4	0	1	0	0	10
UP3	1	3	5	0	1	0	0	10
UP4	7	2	1	0	0	0	0	10
IP1	4	4	2	0	0	0	0	10
IP2	4	2	2	0	2	0	0	10
TOTAL	22	32	31	0	11	3	1	100

Figura 6.2: Estatísticas coletadas dos questionários respondidos

A tabela da Figura 6.2 ilustra que no quesito de clareza e compreensividade o GSE (pergunta FUP1) obteve notas positivas, pois 90% dos voluntários concordaram com algumas ressalvas de que o GSE é claro e compreensível. Em seguida, quando questionados sobre a não exigência de esforço mental para utilizar o GSE (pergunta FUP2), 50% dos voluntários concordaram que a linguagem não demandava muito esforço mental, mas a outra metade da turma discordou com a afirmação de que a linguagem não exige muito do esforço menta do utilizador. Esta afirmação de discordância pode ser verificada pela tabela da Figura 6.2, onde um voluntário marcou "Discordo totalmente"com a pergunta FUP2.

Perguntados, depois, sobre a facilidade de uso do GSE (pergunta FUP3), 80% dos voluntários afirmaram que a linguagem é, de certa forma, fácil de utilizar. Finalizando as perguntas sobre facilidade de uso percebida, foi perguntado se elas conseguiam especificar um sistema utilizando a linguagem (pergunta FUP4). As repostas de 80% dos voluntários foram positivas.

Com estes resultados de facilidade de uso é possível afirmar que o GSE é de fácil uso, pois 75% (30 respostas de concordância de 40 respostas no total) das respostas obtidas na seção do questionário sobre facilidade de uso são positivas. Com estes resultados, passemos para a seção de utilidade percebida (perguntas UP1, UP2, UP3 e UP4).

O fato de os participantes não terem avaliado a linguagem como uma linguagem de fácil uso pode ser ilustrado por um dos comentários feitos por um dos participantes do estudo de caso: "A linguagem especifica bastante, porém em um primeiro momento é difícil de aplicá-la.". Outro participante relata que: "Tive uma certa dificuldade no início, mas depois que fiz a primeira característica, as demais fluíram mais facilmente.". Estes comentários mostram que apesar de os participantes terem acho fácil de usar o GSE no final do estudo de caso, ele tiveram certas dificuldades no início, fazendo com que a nota dado para este quesito na linguagem abaixasse um pouco.

Em utilidade percebida, os voluntários foram perguntados se o desempenho individual dele foi melhorado a partir do uso do GSE (pergunta UP1). A resposta obtida por 90% das pessoas foi de que o GSE melhorou o desempenho de especificação delas, mesmo que pouco. A próxima pergunta explorou o que os peguntados acharam sobre a melhoria na produtividade durante a especificação de sistemas (pergunta UP2). O resultado foi que 90% das pessoas acharam que a produtividade melhorou com o uso do GSE.

A eficácia da especificação produzida também foi considerada melhor para 90% dos voluntários, segundo os resultados da pergunta UP3. No final da seção de utilidade percebida, os voluntários afirmaram com unanimidade que eles consideram a linguagem GSE útil para a especificação de requisitos de sistemas (pergunta UP4).

As respostas obtidas na seção de utilidade percebidas, permitem a conclusão de que as pessoas não envolvidas com o desenvolvimento de software perceberam um avanço na especificação de software quando o GSE é utilizado, pois agora estas pessoas sente-se mais próximas do sistema que querem ver desenvolvido, já que ruídos, antes existentes, na comunicação dos requisitos com os desenvolvedores poderão ser diminuídos.

Alguns dos comentários relacionados com a utilidade percebida seguem os mesmo princípios que este participante afirma: "Acho que vai facilitar os ruídos na comunicação entre cliente e desenvolvedor.". Esta afirmação mostra que mesmo com um pouco de dificuldade no início do estudo de caso, os participantes perceberam um dos pontos chave desta pesquisa: Melhorar a comunicação entre stakeholders e desenvolvedores.

Por fim, o questionário perguntou sobre a intenção de uso futuro caso esta tecnologia esteja disponível. A primeira pergunta deste tópico abordou o fato de o GSE ser considerado na hora de escolher uma tecnologia para especificar sistemas (pergunta IP1). As respostas foram unanimes em afirmar que sim, o GSE seria uma das tecnologias consideradas para especificar sistemas. Por fim, a última pergunta abordou o fato de o GSE ser escolhido para especificar sistemas (pergunta IP2). O resultado foi que 80% dos voluntários afirmou que usariam o GSE para especificar sistemas.

O fato dos usuários considerarem a utilização do GSE para especificar seus sistemas sugere que a linguagem produziu algum valor para os voluntários e de que este valor foi positivo, pois caso contrário eles não gostariam de utilizar novamente esta tecnologia.

O gráfico da Figura 6.3 agrupa as respostas obtidas por seção do questionário, facilitando a visualização das analises realizadas anteriormente.

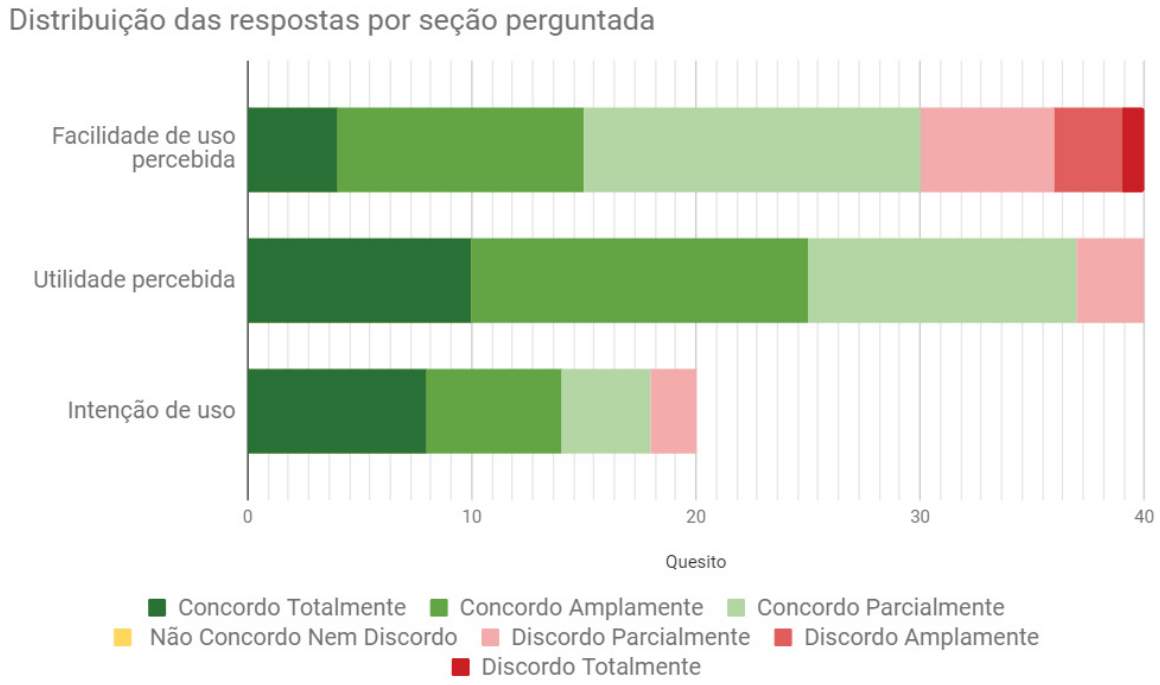
Com o gráfico da Figura 6.3 é possível perceber que a seção de facilidade de uso percebida possui a maioria de suas respostas como sendo *Concordo Parcialmente*. Isso demonstra que, apesar de não ter achado difícil utilizar o GSE, a maioria dos usuários precisariam de mais tempo utilizando o GSE para que eles se sentissem mais confortáveis usando a linguagem. Quando observamos o gráfico referente a utilidade percebida, é possível inferir que a maioria dos usuários percebeu alguma utilidade em utilizar o GSE. Similarmente, o gráfico de intenção de uso sugere que os voluntários voltariam a usar o GSE caso eles precisassem especificar um sistema, isso pode significar que os usuários realmente adotariam a linguagem por ter gostado de usá-la, já que a maioria das respostas desta seção é *Concordo Totalmente*.

Como as alternativas do questionário estão em uma escala baseada na escala de Likert, é possível associar um valor numérico para cada alternativa possível. Estes valores variam entre 0 e 100 como mostrado na tabela da Figura 6.4

Com as notas especificadas na Figura 6.4 atribuídas às respostas foi possível analisar como a linguagem foi avaliada em cada uma das seções: Facilidade de Uso Percebida, Utilidade Percebida e Intenção de Uso. Para tanto foi calculado a nota média de cada seção e o desvio padrão de cada seção, como mostra a Figura 6.5.

As estatísticas da Figura 6.5 permitiram calcular a distribuição normal das notas por seção, cujos gráficos foram plotados e são exibidos na Figura 6.6.

Com o gráfico exibido na Figura 6.6, as médias e o desvio padrão mostrados na Figura 6.5 é possível perceber que os voluntários que participaram do estudo de caso tiveram um alto interesse do GSE, já que a média desta seção foi alta e o desvio padrão não foi alto. Quando analisamos a utilidade percebida, as médias também foram altas, próximas das de intenção de



Quesito	Concordo Totalmente	Concordo Amplamente	Concordo Parcialmente	Não Concordo Nem Discordo	Discordo Parcialmente	Discordo Amplamente	Discordo Totalmente
Facilidade de uso percebida	4	11	15	0	6	3	1
Utilidade percebida	10	15	12	0	3	0	0
Intenção de uso	8	6	4	0	2	0	0

Figura 6.3: Número de respostas de cada tipo por seção do questionário

Resposta	Nota
Concordo Totalmente	100
Concordo Amplamente	83,3
Concordo Parcialmente	66,64
Não Concordo Nem Discordo	50
Discordo Parcialmente	33,33
Discordo Amplamente	16,66
Discordo Totalmente	0

Figura 6.4: Nota de cada resposta no questionário

	Média	Desvio Padrão
Facilidade de Uso Percebida	64,1465	9,464100736
Utilidade Percebida	78,72925	8,616286392
Intenção de Uso	81,651	4,997

Figura 6.5: Média e Desvio Padrão por seção do questionário

uso, contudo, o desvio padrão foi um pouco maior, caracterizando que nem todos os voluntários perceberam a utilidade de forma igual. Da mesma forma podemos analisar a facilidade de uso, que obteve uma média inferior que a de utilidade percebida, e um desvio padrão um pouco maior que o desvio padrão da utilidade percebida.

Analisando agora a especificação criada pelos voluntários através da tabela da Figura 6.7, a qual mostra a média aritmética das especificações dos grupos. As notas variam entre 0 (zero) e

Figura 6.6: Curva normal por seção do questionário

100 (cem). Sendo que 100 é a maior nota possível e 0 é a menor nota possível. Quanto maior a nota, melhor o desempenho do grupo. É importante observar também que as especificações foram feitas por grupo, de forma que cada pessoa do grupo especificou uma funcionalidade do sistema.

	Grupo 1	Grupo 2	Grupo 3
Qualidade da Especificação	88,75	75	65
Uso da seção Característica	100	87,5	50
Uso da seção Descrição	93,25	95,5	66
Uso da seção Restrições e qualidades	79,5	76,25	45
Uso da seção Cenário	78,75	87	67,5
MÉDIA POR GRUPO	88,05	84,25	58,7
MÉDIA TOTAL	77		

Figura 6.7: Estatísticas coletadas das especificações criadas

Analisando a a tabela da Figura 6.7 foi possível perceber que não houveram tantas divergências entre a qualidade das especificações feitas pelos grupos, exceto pelo Grupo 3. É importante observar que o o Grupo 3, o qual teve os valores mais baixos, foi composto de duas pessoas, enquanto que os Grupos 1 e 2 continham quatro pessoas cada. Foi no Grupo 3 que a pessoa que tirou a menor nota estava. Isto fez com que a média deste grupo, que é composto por duas pessoas reduzisse substancialmente.

Entretanto se considerarmos os Grupos 1 e 2, suas notas foram similares, sendo todas acima de 75. Isso permite a inferência de que mesmo em apenas um contato pessoas não experientes com a especificação de sistemas puderam especificar com uma boa qualidade. Ainda sobre a Figura 6.7, é possível perceber que a maior dificuldade dos voluntários foi com a especificação de Restrições e qualidades (*Constraints and qualities*). Há algumas observações para tal fato.

A primeira observação é que esta seção, restrições e qualidades, é a que contém mais detalhes gramaticais em relação às outras seções apresentadas para os voluntários. Outra observação é a de que a seção de restrições e qualidades é a responsável por especificar os requisitos não-funcionais, portanto é natural que pessoas que nunca se depararam com a especificação de sistemas tenham dificuldades em detalhar requisitos não funcionais, já que os detalhes de implementação as vezes podem não ser óbvios.

A terceira observação é que a gramática definida para seção restrições e qualidades exige que o usuário pense e redija de uma forma não muito natural. Isso porque é exigido pela gramática a especificação do que o ator não pode ser capaz de fazer, e o que o requisito deve implementar. A gramática foi definida desta forma porque pela definição de requisitos não funcionais (Widrig e Leffingwell, 1999) (Farrell e Truitt, 2004), os requisitos não funcionais podem ser considerados como: O que uma funcionalidade deve implementar para funcionar e as restrições que são impostas aos atores do requisito. A mudança de paradigma que o estudo de caso explorou, de passar de um usuário que apenas usa um sistema para um usuário capaz de especificar um sistema, pode demandar mais esforço cognitivo e prática. Portanto é natural que a seção de restrições e qualidades tenha a menor nota entre as seções apresentadas.

6.3.2 Turma PET Informática

Realizando as mesmas análises realizadas com os dados obtidos com a turma de Tecnologia em Comunicação Institucional, será, inicialmente avaliado as respostas do questionário e, em seguida, a qualidade da especificação gerada.

Começando com a análise das respostas do questionário, é possível criar o gráfico da Figura 6.8. Nele é possível ver o número de cada tipo de resposta por pergunta.

ID da Pergunta	Número de Concordo Totalmente	Número de Concordo Amplamente	Número de Concordo Parcialmente	Número de Não Concordo Nem Discordo	Número de Discordo Parcialmente	Número de Discordo Amplamente	Número de Discordo Totalmente	Número de Respostas
FUP1	0	8	1	1	0	0	0	10
FUP2	2	3	4	1	0	0	0	10
FUP3	4	4	2	0	0	0	0	10
FUP4	4	3	3	0	0	0	0	10
UP1	2	3	1	4	0	0	0	10
UP2	2	4	1	3	0	0	0	10
UP3	2	3	2	3	0	0	0	10
UP4	7	3	0	0	0	0	0	10
IP1	2	7	1	0	0	0	0	10
IP2	3	4	2	1	0	0	0	10
TOTAL	28	42	17	13	0	0	0	100

Figura 6.8: Estatísticas coletadas dos questionários aplicados ao pessoal do PET Informática

Analisando a tabela da Figura 6.8, é possível perceber que 90% dos voluntários afirmaram ter uma interação relativamente clara e compreensível com o GSE (pergunta FUP1), sendo que 10% dos voluntários tiveram um pouco mais de dificuldade em compreender a linguagem. Em seguida, a pergunta feita compreendida a não exigência de esforço mental dos voluntários (pergunta FUP2). Nesta pergunta, novamente, 90% dos participantes afirmaram não ter precisado

empregar muito esforço mental para utilizar a linguagem, enquanto que 10% precisou empregar um pouco mais de esforço.

Quando questionados se a linguagem é fácil de usar ou não (pergunta FUP3), os participantes responderam, em unanimidade, que a linguagem é fácil, mesmo que algum aspectos dela precisem de um pouco mais de prática. Os voluntários afirmaram com unanimidade que o GSE é fácil de usar para especificar sistemas de forma que todos os stakeholders consigam entender (pergunta FUP4).

Estas estatísticas que compõe a seção de facilidade de uso são validadas por comentários dos participantes, dentre eles pode-se citar um participante que disse: "A linguagem possui uma divisão bastante clara e objetiva". Afirmações similares a citada são comum de encontrar nos questionários de outros participantes do PET.

A pergunta seguinte questionou o desempenho do usuário na especificação de requisitos com o GSE (pergunta UP1). Nesta pergunta 40% dos voluntários afirmaram que não sabem se houve ou não melhora, enquanto que os outros 60% disse que houve melhora no desempenho. Esta resposta é esperada, pois como a boa parte dos voluntários são estudantes de início do curso, eles podem não ter experienciado ainda o fato de precisar especificar algo. Se o experienciaram, podem não ter prestado atenção para ter dados o suficiente para comparar com a experiência atual com o GSE.

Uma conclusão similar pode ser obtida da questão que perguntou se a produtividade foi aumentada (questão UP2). Nessa última pergunta, 30% dos participantes responderam não saber se a produtividade foi melhorada ou não com o uso do GSE, enquanto que 70% disseram que houve um aumento na produtividade quando começaram usar o GSE. Seguindo um padrão similar, pelo motivo semelhante, ao da questão anterior, a pergunta seguinte, que questionou sobre a eficácia dos participantes na especificação de sistema (pergunta UP3), obteve 30% do participantes afirmando não saber se a eficácia da especificação foi melhorada, enquanto que 70% dos participantes afirmaram que sentiram uma melhora na eficácia na especificação de requisitos.

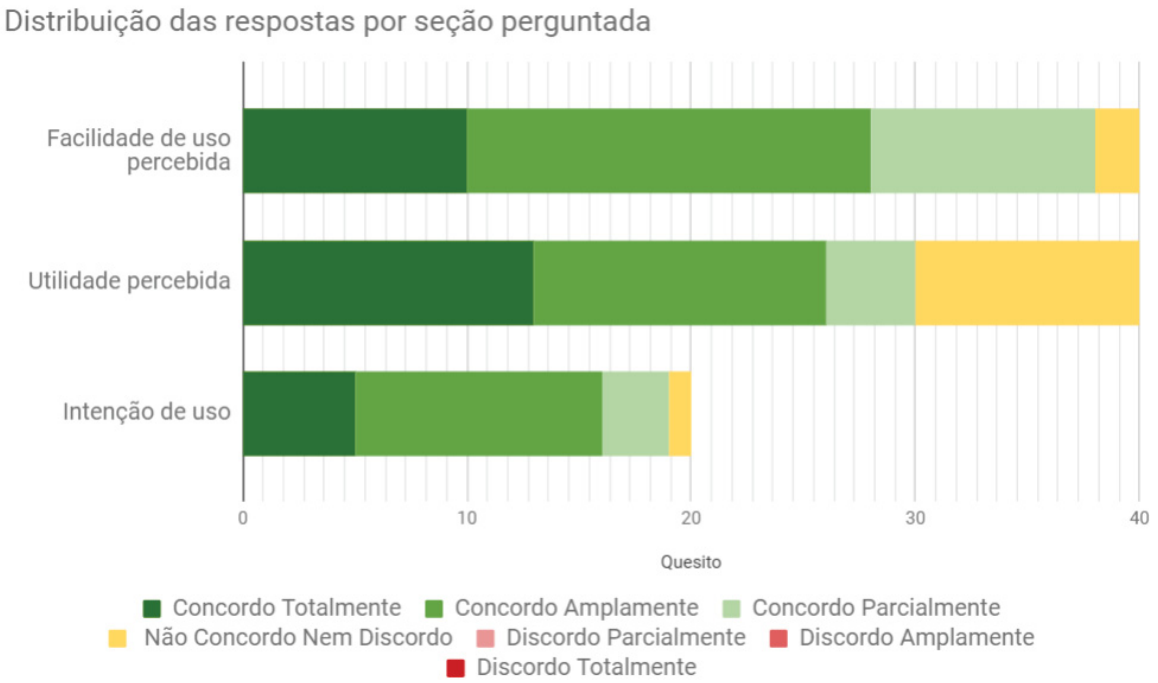
A questão seguinte, que perguntou se o participante considera a linguagem GSE útil na especificação de requisitos (pergunta UP4), foi considerada, com 100% das respostas, como um linguagem útil.

As afirmações do participantes relacionadas à utilidade percebida focam principalmente na padronização das informações, como outro participante afirma: "É muito útil, padroniza as informações para um melhor entendimento do sistema. Poupa tempo e dúvidas do desenvolvedor", outro participante diz: "A linguagem aparenta ser bastante útil como primeiro acesso de modelagem pela facilidade de seu uso". Os participantes que tiveram mais dificuldades não deixaram comentários a respeito de suas dificuldades.

Na pergunta que questionou se o participante pretenderia usar o GSE caso ele tivesse acesso a ele (pergunta IP1), teve como resposta de todos os voluntários uma afirmativa. Por fim, a última questão perguntou se o participante prevê ele usando a linguagem (pergunta IP2). Nesta questão, 10% dos voluntários afirmaram não saber se usariam a linguagem, enquanto que 90% dos voluntários disseram que preveem o uso do GSE.

Os comentários sobre a intensão de uso futuro seguem a linha de que os participantes gostariam de utilizar a linguagem no futuro. Como diz um dos participantes, "Pela curta experiência de uso, me interessei pela linguagem e acredito que usaria de novo em um projeto completo como teste". Entretanto mais de um participante cita uma possível dificuldade em mudar os padrões já estabelecidos, como diz este outro participante: "É difícil mudar os métodos que já utilizamos, por isso acho que seria um pouco puxado começar a utilizá-la [utilizar o GSE] totalmente. Para mim é benéfico, para alguns casos específicos. Considero um linguagem promissora".

Para analisar as respostas do ponto de vista da seção do questionário que a pergunta faz parte, o gráfico da Figura 6.9 foi criado.



Quesito	Concordo Totalmente	Concordo Amplamente	Concordo Parcialmente	Não Concordo Nem Discordo	Discordo Parcialmente	Discordo Amplamente	Discordo Totalmente
Facilidade de uso percebida	10	18	10	2	0	0	0
Utilidade percebida	13	13	4	10	0	0	0
Intenção de uso	5	11	3	1	0	0	0

Figura 6.9: Gráfico mostrando o número de respostas de cada tipo por seção - PET

O gráfico da Figura 6.9 mostra de forma mais clara que o fato de o público ter pouca experiência com especificações faz com que alguns participantes não tenham certeza se o GSE é ou não útil. Quanto à facilidade de uso, os participantes não tiveram tanta dificuldade em utilizar a linguagem. O mesmo pode ser dito da intenção de uso: os usuários se tiverem oportunidade provavelmente usarão o GSE para especificar requisitos.

A tabela apresentada na Figura 6.10 mostra a nota média e o desvio padrão que cada seção da linguagem recebeu através do questionário.

	Média	Desvio Padrão
Facilidade de Uso Percebida	81,645	4,251995414
Utilidade Percebida	78,7365	9,457529633
Intenção de Uso	83,311	1,663

Figura 6.10: Média e Desvio Padrão por seção do questionário - PET

Estes números, mostrados na Figura 6.10 são utilizados pra gerar as curvas da distribuição normal de cada seção, disponível na Figura 6.11. Estas curvas permitem a visualização um pouco mais clara dos dados expostos até este então nesta Subseção.

A Figura 6.11 mostra claramente que a utilidade percebida tem a maior variância nas respostas, pois esta foi a seção que mais obteve respostas *"não concordo nem discordo"*. Intenção

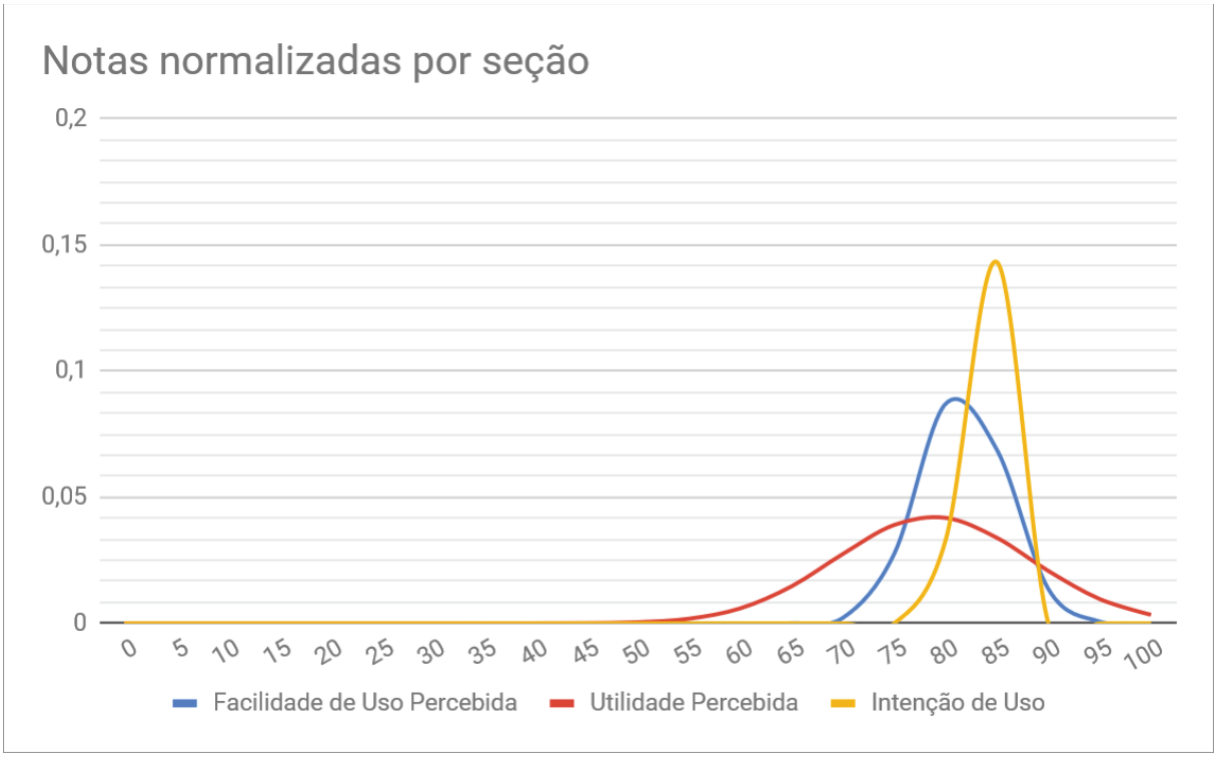


Figura 6.11: Curva normal por seção do questionário - PET

de uso, como já discutido, é a que possui a menor variância nas respostas, com a maior parte dos voluntários afirmando que usariam o GSE. Por fim, a facilidade de uso percebida ficou na média, com alguns usuários com um pouco mais dificuldade que outros para entender e utilizar o GSE. Entretanto, a maior parte das respostas conduzem para a conclusão de que o GSE não é difícil de usar, apenas exige um pouco de prática.

Quando avaliamos a qualidade das especificações criadas pelos participantes, temos como resposta a tabela da Figura 6.12. Nesta Figura, é possível reparar que as especificações criadas possuem uma boa qualidade, mesmo com notas não tão boas nas seções da linguagem. Estas notas mais baixas nas seções da linguagem são, em sua maioria, devido a inadequações no uso da gramática do GSE e não ao conteúdo das seções.

	Grupo 1	Grupo 2	Grupo 3
Qualidade da Especificação	79,25	70	72,66666667
Uso da seção Característica	100	100	81
Uso da seção Descrição	69,25	65,33333333	36
Uso da seção Restrições e qualidades	68,25	62,33333333	50
Uso da seção Cenário	64,75	33,33333333	96
MÉDIA POR GRUPO	76,3	66,2	67,13333333
MÉDIA TOTAL	**69,87777778**		

Figura 6.12: Estatísticas coletadas das especificações criadas - PET

De um maneira geral, o uso da linguagem manteve-se consistente entre os grupos, o grupo 1 manteve uma consistência maior no uso das diversas seções do GSE, enquanto que o grupo 2 teve mais dificuldade especificando os cenários e o grupo 3 teve mais dificuldade especificando os cenários e as restrições e qualidades.

6.3.3 Discussão dos resultados

Quando comparamos as curvas normais das Figuras 6.6 e 6.11 percebemos que os participantes do curso de Comunicação Institucional teve uma maior variância nas respostas fornecidas no quesito facilidade de uso percebida. Por este motivo, a curva que representa esta característica é mais aberta que a curva dos participantes do PET.

Este resultado pode ter acontecido porque, mesmo sendo estudantes de início de curso, os voluntários do PET Informática já tem começado a aprender formas mais eficazes de aprender novas linguagens, já que os cursos de Ciência da Computação e Informática Biomédica ensinam desde o início seus estudantes a aprender linguagens não naturais. Enquanto isso, os estudantes de Comunicação Institucional, que não tem contato com linguagens não naturais em sua grade curricular, podem ter tido seu primeiro contato com uma linguagem que não é linguagem natural.

A avaliação de utilidade percebida foi similar entre as duas turmas. Isso pode ter acontecido porque ambas as turmas perceberam algo na linguagem que a acharam útil. Tanto que ambas as curvas tem como nota média um valor próximo de 80, onde a nota máxima é 100.

A avaliação em que houve a maior diferença entre as turmas foi na avaliação de intenção de uso. A diferença é que turma de comunicação institucional teve uma maior variância maior que a turma do PET. A hipótese é que este fato aconteceu porque o pessoal do PET informática sabe que um dia terá que trabalhar com especificação de requisitos de sistema e, como eles podem não conhecer outra forma de especificar requisitos, eles consideram o GSE como a primeira ferramenta de especificação de requisitos quando for preciso especificar requisitos. Enquanto que os participantes de comunicação institucional podem não ter encontrado um uso para o GSE de imediato, fazendo com que alguns participantes marcassem como não tendo tanta intenção de uso quanto o pessoal do PET.

6.4 AMEAÇAS À VALIDADE

Como explicado na Seção 6.1.3, para que a validação da linguagem tivesse uma maior confiabilidade, a linguagem foi traduzida sem perda alguma de semântica. Entretanto, esta tradução não permitiu que o interpretador da linguagem criada fosse validado, já que este interpretador foi desenvolvido para a versão original em inglês. Embora não impacte na avaliação da linguagem, o interpretador avaliaria a geração automática de documentação através de requisitos criados por *stakeholders* não desenvolvedores. Esta documentação também poderia ser avaliada pelos participantes dos estudos de casos. Criar o suporte para outros idiomas está como trabalhos futuros do autor, portanto não foi possível processar os dados em português criados pelos participantes do estudo de caso no interpretador já implementado.

Outra ameça à validade é o fato de terem sidos executados apenas dois estudos de casos pequenos. Para ter uma mair precisão no resultado mais estudos de caso deveriam ser executados com mais participantes. O fato de a maior parte do público que participou dos estudos de casos ser estudantes também é uma ameaça à validade deste estudo.

Outra ameça identificada é a não avaliação da linguagem por especialistas em engenharia de requisitos. Estes especialistas poderiam dar uma avaliação mais completa e precisa sobre os pontos fortes e fracos do GSE.

6.5 PERCEPÇÕES OBTIDAS DURANTE OS ESTUDOS DE CASOS

Durante os estudos de casos, foi observado que os participantes tiveram uma certa dificuldade em entender como uma especificação de sistema é feita, mesmo tendo entendido uma especificação após lê-la.

Além disso, outra dificuldade apresentada por alguns voluntários foi a compreensão de que cada conjunto de seções do GSE define uma funcionalidade de um sistema. Para alguns voluntários o sistema inteiro deveria ser definido por apenas um conjunto de seções do GSE.

No geral, a seção de Restrições e Qualidades foi a que o autor mais percebeu dificuldade de entendimento pelos participantes. Em contrapartida, a seção Descrição foi a seção que os voluntários tiveram a maior facilidade em entender. Estes fatos podem ser comprovados pelas especificações criadas por eles, as quais foram discutidas na Seção 6.3 .

Alguns voluntários no final do estudo de caso elogiaram a iniciativa de tentar criar uma linguagem que melhore a comunicação entre desenvolvedores e clientes. Como respondido no questionário, eles acharam o GSE útil, mas reportaram pessoalmente que precisariam, para serem capazes de especificar, um pouco mais de prática. Esta é outra afirmação que pode ser analisada com os resultados dos estudos de casos, discutidos anteriormente na Seção 6.3.

6.6 CONCLUSÃO

Para avaliar se as premissas da linguagem são atingidas, isto é, se a linguagem consegue permitir a comunicação entre usuários não técnicos e desenvolvedores sem exigir que os usuários não técnicos precisem aprender muitos conceitos novos, foram conduzido dois estudos de casos com duas turmas distintas, a turma que estuda comunicação institucional e a turma do PET Informática.

Quando os dados fornecidos pelo questionário são analisados, é obtido uma boa taxa de concordância em relação a utilização da linguagem GSE, como é mostrado na Figura 6.13, já que a menor taxa de concordância é de 75%. Nesta figura, foram agrupadas as respostas fornecidas pelo questionário, de forma que todas as opções de concordância do questionário formou a Opinião Concordo, e todas as opções de discordância formou a Opinião Não concorda. Então foram calculados os valores percentuais de cada opinião em relação a cada seção do questionário.

De uma maneira geral, com os resultados obtidos com estes estudo de casos, é possível afirmar que um dos principais requisitos do GSE foi alcançado, o fato de ele ser de fácil uso para usuários que não estão inseridos no universo de desenvolvimento de software. Isto foi ilustrado através das repostas obtidas no questionário aplicado, as quais foram sumarizadas na tabela da Figura 6.13, o qual mostrou que os participantes tiveram um relativa facilidade no uso do GSE, percebem a utilidade da ferramenta e pretendem voltar utilizá-la.

Os dois estudos de casos realizados mostram que mesmo utilizado por grupos diferentes os resultados são semelhantes, tanto na qualidade da especificação quanto nas notas dos questionários. Estes fatos podem ser observados a partir da comparação dos resultados obtidos tanto do questionário quanto da qualidade da especificação produzida. A qualidade da especificação gerada pode ser ilustrada pela nota média de cada turma. A nota média da turma de comunicação institucional é 77, enquanto que a nota média da turma do PET Informática é 69,8778. A proximidade destes valores permite a conclusão de que a linguagem consegue cumprir de forma satisfatória sua premissa.

Complementando o questionário, a qualidade das especificações desenvolvidas mostraram-se promissoras, com pequenos erros e detalhes que poderiam ser reduzidos consideravelmente se os voluntários utilizassem a ferramenta por mais tempo, uma vez que os erros cometidos são pela falta de experiência em especificação de sistemas, ou por detalhes mínimos de utilização da gramática corretamente. Estes erros na realidade não comprometem tanto a proposta do GSE porque geralmente quem irá especificar o software será alguém experiente em especificações, o papel do leigo em especificações será a de validar o sistema, e para especificar ele terá que ler uma especificação. Como os voluntários conseguiram até escrever especificações

Quesito	Opinião	Comunicação Institucional	PET Informática
Facilidade de uso	Concorda	75,00%	95,00%
	Não concorda nem discorda	0,00%	5,00%
	Não concorda	25,00%	0,00%
Utilidade de uso	Concorda	92,50%	75,00%
	Não concorda nem discorda	0,00%	25,00%
	Não concorda	7,50%	0,00%
Intenção de uso	Concorda	90,00%	95,00%
	Não concorda nem discorda	0,00%	5,00%
	Não concorda	10,00%	0,00%

Figura 6.13: Comparação entre as notas dos estudo de casos

com o GSE, é plausível inferir que os voluntários são capazes de ler e entender uma especificação escrita com o GSE, já que para ser capaz de escrever, primeiro é preciso saber ler.

7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o Gherkin Specification Extension (GSE), uma extensão da já existente linguagem de definição de testes Gherkin. Para seu desenvolvimento foram executadas várias etapas. A primeira delas foi a verificação de um problema que pudesse ser abordado. O problema encontrado foi a da comunicação entre stakeholders em todo o processo de desenvolvimento, em especial na definição dos requisitos do sistema. Com este problema de pesquisa, foi realizado uma pesquisa do estado da arte atual, a qual verificou que pouco havia sido feito para melhorar a comunicação entre clientes, usuários, desenvolvedores e analistas de negócios.

Para tentar solucionar este problema, foi estudado formas de mitigar o problema utilizando tecnologias já existentes. Quando o Gherkin foi descoberto, foi estudado uma forma de melhorar seu poder de descrição para que ele pudesse ser utilizado não apenas para definir testes, mas também especificar requisitos. Após este estudo, uma lista com, inicialmente, treze requisitos surgiu e uma primeira versão da extensão foi desenvolvida. Após analisar os requisitos definidos e a extensão desenvolvida, percebeu-se que haviam espaços para melhorias. Então os onze requisitos em vigor surgiram como aperfeiçoamento dos treze requisitos iniciais. Junto com os novos requisitos, uma nova versão da extensão foi criada.

Esta nova versão foi utilizada para criar o interpretador da linguagem contemplando, todos os requisitos que dependem de um interpretador para serem implementados. Embora o autor acreditasse que qualquer pessoa que tivesse contato com o GSE conseguisse entender, mesmo que com certa dificuldade, o que estava sendo especificado, era preciso avaliar se esta suspeita era verdadeira. Então foi pesquisado métodos de avaliar esta suspeita, sendo o Technology Acceptance Model o modelo que melhor capturava os pontos que o autor queria avaliar: Facilidade de uso, Utilidade percebida e Intenção de uso futuro. Então, em dois dias distintos, foram realizados os estudos de casos para avaliar a suspeita do autor. Os estudos de casos foi composto de um aula inicial para explicar como utilizar o GSE, um exercício para que os voluntários pudessem aplicar o que foi explicado e posteriormente o questionário baseado no TAM foi aplicado.

Os resultados dos estudos de casos na avaliação do TAM na parte de facilidade de uso foi de 75% e 95% de concordância, no quesito utilidade de uso foi obtida 92,50% e 75% de concordância e em intenção de uso foi obtido 90% e 95% de concordância. Quando avaliada a qualidade da especificação gerada, o autor deste trabalho obteve uma média de 77 para um grupo e 69,87 para o outro grupo de um total de 100.

Os resultados obtidos indicaram que os *stakeholders* não técnicos, embora com dificuldade no início dos estudos de casos, tiveram uma recepção positiva quanto ao GSE. Isto ocorreu devido ao fato que os participantes não apenas leram especificações escritas usando o GSE, eles escreveram especificações utilizando o GSE. O fato que mais gerou complicações na utilização do GSE foi ter uma gramática que restringia a formação de sentenças e estruturas que são comumente utilizadas nas linguagens naturais, como por exemplo a definição de um requisito não-funcional, onde deve-se escrever o que o ator não deve ser capaz de fazer e o que o requisito deve implementar.

As especificações geradas pelos participantes nos estudos de casos e avaliadas pelo autor deste trabalho seguindo os critérios apresentados nas Seções 6.3.1 e 6.3.2 foram, em sua maioria, criadas com uma boa qualidade, o que indica que com um pouco mais de prática os stakeholders alvo desta pesquisa conseguiriam utilizar perfeitamente a tecnologia desenvolvida neste trabalho.

O fato de que os voluntários conseguiram utilizar o GSE gera vários pontos positivos para o processo de desenvolvimento de software. Isto porque logo na primeira etapa de desenvolvimento os testes de aceitação já estarão sendo definidos para que posteriormente sejam executados. Isto ocorre porque com a utilização do GSE e do Gherkin, além de definir melhor os requisitos, os testes também são definidos.

Adicionalmente, a geração automática de documentação, criada a partir do interpretador do GSE criada neste trabalho, também auxilia uma equipe de desenvolvimento fazendo com que menos tempo seja dispensado para a escrita da documentação do projeto, permitindo que os desenvolvedores tenham seu foco apenas em criar o sistema almejado pelo cliente.

Complementarmente, o interpretador do GSE permite a exportação dos requisitos em formatos (CSV e JSON) que são mais facilmente consumidos em algoritmos, permitindo que pesquisadores e equipes de desenvolvimento possam utilizar estes dados em algoritmos de inteligência artificial ou processá-los para obter informações de *Buissness Inteligence* (BI). Com algoritmos de inteligência artificial, será possível, por exemplo, agrupar requisitos semelhantes automaticamente. No caso do BI, processar os requisitos para obter dados conseguirá, por exemplo, informações do tipo: quais são os atores mais importantes no sistema, quantas funcionalidades cada ator possui em média, entre outras informações.

Como trabalhos futuros, será trabalhada a possibilidade de aumentar o número de idiomas suportados para o GSE para as mesmas linguagens suportadas pelo Gherkin. Desta forma será mais fácil a utilização do GSE por diferentes pesquisadores e equipes de desenvolvimento. Há ainda a ideia de implementar uma forma mais natural e prática de definir os requisitos não-funcionais. Atualmente, os requisitos não-funcionais são suportados pela sua escrita em todas as funcionalidades que deverão implementar estes requisitos. Caso um requisito não-funcional deva ser implementado no sistema inteiro, todos os arquivos contendo os requisitos funcionais deverão conter sua especificação. Algo que pode ser inconveniente, já que ter que escrever a mesma frase em muitos arquivos diferentes pode ser tedioso e causar erros caso quem especifica perca o foco por um momento.

REFERÊNCIAS

Abran, A., Moore, J. W., Bourque, P. e Dupuis, R. (2004). *Guide to the Software Engineering Body of Knowledge (SWEBOK)*.

Adzic, G. (2011). *Specification by Example How successful teams deliver the right software*. Manning Publications Co.

Ali, N. A., Mirghani, A. A., Ibrahim, A. Y. e Madani, W. (2017). Alneelain: A Formal Specification Language. *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*.

Anggraini, R. N. e Martin, T. P. (2018). Fuzzy Representation for Flexible Requirement Satisfaction. *Advances in Intelligent Systems and Computing*, 650:28–36.

Aslak Hellesøy, Joseph Wilk, M. W. G. H. M. S. (2008). Cucumber documentation. `https://cucumber.io/docs`. Acessado em 02/03/2018.

Attiogbé, J. C. (2018). Formal software engineering - lecture notes. `http://pagesperso.ls2n.fr/~attiogbe-c/mespages/MSFORMEL/cours_B_ALMA_2018.2x1.pdf`. Acessado em 06/12/2018.

Aysolmaz, B., Leopold, H., Reijers, H. A. e Demirörs, O. (2018). A semi-automated approach for generating natural language requirements documents based on business process models. *Information and Software Technology*, 93:14–29.

B. Nuseibeh, S. M. E. (2007). *Fundamentals of Requirements Engineering*. Pearson Education.

Bäumer, F. S. e Geierhos, M. (2016). Running out of words: How similar user stories can help to elaborate individual natural language requirement descriptions. Em *Communications in Computer and Information Science*, volume 639, páginas 549–558.

Bolic, I. (2017). Gherkin on github. `https://github.com/cucumber/cucumber/wiki/Gherkin`. Acessado em 27/02/2018.

Casamayor, A., Godoy, D. e Campo, M. (2012). Functional grouping of natural language requirements for assistance in architectural software design. *Knowledge-Based Systems*.

Costello, R. J. e Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems and Software*, 29(1):39 – 63. Oregon Metric Workshop.

da Silva, A. A. R., Savić, D., Vlajić, S., Antović, I., Lazarević, S., Stanojević, V., Milić, M., Savic, D., Vlajic, S., Antovic, I., Lazarevic, S., Stanojevic, V. e Milic, M. (2015). A pattern language for use cases specification. *Proceedings of the 20th European Conference on Pattern Languages of Programs*, 08-12-July(July 2015):8.

da Silva Xavier, C. M. (2014). Como coletar os requisitos em projetos. `https://beware.com.br/academia/artigos/coletar-requisitos/`. Acessado em 08/03/2019.

Daniel Jackson, Aleksandar Milicevic, E. T. E. K. J. N. (2017). About alloy. `http://alloytools.org/about.html`. Acessado em 12/03/2018.

Dardenne, A., van Lamsweerde, A. e Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50.

David Astels, Granville Miller, M. N. (2002). *Extreme Programming Guia Prático*. Campus.

Davies, J. e Woodcock, J. (1996). *Using Z: Specification, Refinementm and Proof*. In Prentice Hall.

De, D., Ferreira, A., Rodrigues, A. e Silva, D. (2012). RSLingo: An Information Extraction Approach toward Formal Requirements Specifications. *Model-Driven Requirements Engineering Workshop (MoDRE), 2012 IEEE*.

De Almeida Ferreira, D. e Da Silva, A. R. (2013). RSL-IL: An interlingua for formally documenting requirements. Em *2013 3rd International Workshop on Model-Driven Requirements Engineering, MoDRE 2013 - Proceedings*.

Desai, A., Gulwani, S., Hingorani, V., Jain, N., Karkare, A., Marron, M., R, S. e Roy, S. (2015). Program Synthesis using Natural Language. *Proceeding ICSE '16 Proceedings of the 38th International Conference on Software Engineering*.

Dwivedi, A. K. e Rath, S. K. (2012). Model to specify real time system using Z and Alloy languages: A comparative approach. *IET Seminar Digest*, 2012(4).

El Ahmar, Y., Gerard, S., Dumoulin, C. e Le Pallec, X. (2015). Enhancing the communication value of UML models with graphical layers. *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, MODELS 2015 - Proceedings*, páginas 64–69.

Farrell, K. T. e Truitt, M. (2004). Defining functional requirements for acquisitions records: Vendor metadata. *Library Collections, Acquisition and Technical Services*, 28(4):473–487.

Figueroa-Martinez, J., López-Jaquero, V., Gutiérrez Vela, F. L. e González, P. (2013). Enriching UsiXML language to support awareness requirements. Em *Science of Computer Programming*.

Fox, S. (2016). Behavior driven. `http://behaviour-driven.org/need-know-behaviour-driven-software.html`. Acessado em 02/03/2018.

Freeman, R. E. (1984). *Strategic Management: A Stakeholder Approach*. Boston MA: Pitman.

Freeman, R. E. (2004). *A Stakeholder Theory of Modern Corporations*. Ethical Theory and Business, 7th edition edition.

Friedman, A. L. e Miles, S. (2006). *Stakeholders: Theory and Practice*. Oxford University Press.

Haidrar, S., Bencharqui, H., Anwar, A., Bruel, J. M. e Roudies, O. (2017). REQDL: A requirements description language to support requirements traces generation. Em *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*.

Hellesøy, A. (2018). Cucumber team. `https://docs.cucumber.io/team/`. Acessado em 08/03/2019.

Helm, R. e Wildt, D. (2014). *Histórias de Usuário - Por que e como escrever requisitos de forma ágil?* `http://www.wildtech.com.br/historias-de-usuario/`.

Hesenius, M., Griebe, T. e Gruhn, V. (2014). Towards a behavior-oriented specification and testing language for multimodal applications. *EICS 2014 - Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, páginas 117–122.

K. Beck, M. Beedle, A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. D. T. (2001). Princípios por trás do manifesto Ágil. `http://agilemanifesto.org/iso/ptbr/principles.html`. Acessado em 18/01/2019.

Karu, M. (2013). A textual domain specific language for user interface modelling. Em *Lecture Notes in Electrical Engineering*.

Kaur, A., Gulati, S. e Singh, S. (2012). A comparative study of two formal specification languages: Z-notation & B-method. *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, páginas 524–530.

Klastorin, T. (2004). *Project Management Tools and Trade-Offs*, páginas 83–164. Wiley.

Lago, P., Koçak, S. A., Crnkovic, I. e Penzenstadler, B. (2015). Framing sustainability as a property of software quality. *Communications of the ACM*, 58(10):70–78.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55.

Liu, D., Zhu, H. e Bayley, I. (2014). SOFIA: An algebraic specification language for developing services. Em *Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014*.

Mahmud, N., Seceleanu, C. e Ljungkrantz, O. (2015). ReSA: An ontology-based requirement specification language tailored to automotive systems. Em *2015 10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015 - Proceedings*.

Malan, R., Bredemeyer, D. e Consulting, B. (2001). Defining Non-Functional Requirements. Relatório técnico.

McDonald, K. J. (2016). User story - glossary. `https://www.agilealliance.org/glossary/user-stories`. Acessado em 01/03/2018.

Nami, M. R. e Hassani, F. (2009). A comparative evaluation of the Z, CSP, RSL, and VDM languages. *ACM SIGSOFT Software Engineering Notes*, 34(3):1–4.

Navarro-Almanza, R., Juurez-Ramirez, R. e Licea, G. (2018). Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification. Em *Proceedings - 2017 5th International Conference in Software Engineering Research and Innovation, CONISOFT 2017*.

Nuseibeh, B. e Easterbrook, S. (2000). Requirements engineering: a roadmap. Em *Proceedings of the conference on The future of Software engineering - ICSE '00, 4-11 June 2000*, volume 1, páginas 35–46.

Oda, T., Araki, K. e Larsen, P. G. (2015). VDMPad: A Lightweight IDE for Exploratory VDM-SL Specification. Em *Proceedings - 3rd FME Workshop on Formal Methods in Software Engineering, Formalise 2015*.

Overbeek, S., Frank, U. e Köhling, C. (2015). A language for multi-perspective goal modelling: Challenges, requirements and solutions. *Computer Standards and Interfaces*.

Rauterberg, M. (1996). How to Measure Cognitive Complexity in Human-Computer Interaction. *Cybernetics and Systems*, páginas 815–821.

Respect-IT (2007). A kaos tutorial. `http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf`. Acessado em 15/03/2018.

S. Friedenthal, A. M. e Steiner, R. (2012). *A Practical Guide to SysML The Systems Modeling Language 2nd edition*, páginas 318–342. Elsevier.

Savić, D., Da Silva, A. R., Vlajić, S., Lazarević, S., Antović, I., Stanojević, V. e Milić, M. (2014). Preliminary experience using JetBrains MPS to implement a requirements specification language. Em *Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014*.

Schraps, M. e Peters, M. (2014). Semantic annotation of a formal grammar by SemanticPatterns. Em *2014 IEEE 4th International Workshop on Requirements Patterns, RePa 2014 - Proceedings*.

Serna, J., Day, N. A. e Farheen, S. (2017). DASH: A new language for declarative behavioural requirements with control state hierarchy. Em *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, páginas 64–68.

Smialek, M. e Straszak, T. (2012). Facilitating transition from requirements to code with the ReDSeeDS tool. Em *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*.

Sommerville, I. (2011). *Software Engineering*. Pearson, 9th edition edition.

Stoel, J., van der Storm, T., Vinju, J. e Bosman, J. (2016). Solving the bank with Rebel: on the design of the Rebel specification language and its application inside a bank. *Proceedings of the 1st Industry Track on Software Language Engineering - ITSLE 2016*, 3:13–20.

Subburaj, V. H. e Urban, J. E. (2013). A Formal Specification Language For Modeling Agent Systems. *2013 Second International Conference on Informatics & Applications (ICIA)*, páginas 300–305.

Teruel, M. A., Navarro, E., López-Jaquero, V., Montero, F., Jaen, J. e González, P. (2012). Analyzing the understandability of Requirements Engineering languages for CSCW systems: A family of experiments. *Information and Software Technology*.

ur Rehman, T., Khan, M. N. A. e Riaz, N. (2013). Analysis of Requirement Engineering Processes, Tools/Techniques and Methodologies. *International Journal of Information Technology and Computer Science*, 5(3):40–48.

Venkatesh, V. e Bala, H. (2008). Technology acceptance model 3 and a research agenda on interventions. *Decision Sciences*.

Vidya Sagar, V. B. R. e Abirami, S. (2014). Conceptual modeling of natural language functional requirements. *Journal of Systems and Software*.

Vilela, J., Castro, J., Eduardo Martins, L. G., Gorschek, T. e Silva, C. (2017). Specifying Safety Requirements with GORE languages. *Proceeding SBES'17 Proceedings of the 31st Brazilian Symposium on Software Engineering*, páginas 154–163.

Wang, Y., Zhao, L., Wang, X., Yang, X. e Supakkul, S. (2013). PLANT: A pattern language for transforming scenarios into requirements models. *International Journal of Human Computer Studies*.

Widrig, D. e Leffingwell, D. (1999). *Managing Software Requirements: A Unified Approach*. Addison-Wesley Professional.

Xie, H., Chang, C. K., Ming, H. e Lu, K. S. (2012). The concepts and ontology of SiSL: A situation-centric specification language. Em *Proceedings - International Computer Software and Applications Conference*.

Yu, E. S. (2009). Social modeling and i*. Em *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5600 LNCS, páginas 99–121.

Zhang, H., Yue, T., Ali, S., Wu, J. e Liu, C. (2017). A Restricted Natural Language Based Use Case Modeling Methodology for Real-time Systems. Em *Proceedings of the 9th International Workshop on Modelling in Software Engineering*, páginas 5–11.

APÊNDICE A – APÊNDICE I

O apêndice I contem os arquivos complementares. O número de páginas e a ordem em que os arquivos aparecem estão listados a seguir:

A.1. Questionário pós-uso GSE - 2 páginas

A.2. Solução do Calendário Setorial - 4 páginas

A.3. Apresentação utilizada no treinamento do experimento - 6 páginas

A.4. Gramática GSE - 3 páginas

A.1 QUESTIONÁRIO DE AVALIAÇÃO

A seguir encontra-se o questionário pós-uso utilizado durante o experimento. Ele foi criado com base no Technology Acceptance Model 3.

LINGUAGEM GSE – Questionário pós-uso

Participante:	

Responda as questões a seguir considerando sua experiência durante a inspeção utilizando a **Linguagem GSE**:

1. Em relação à sua percepção sobre a **Facilidade de Uso da Linguagem GSE**, qual o seu grau de concordância em relação às seguintes afirmações (marque com um X):

	Concordo Totalmente	Concordo Amplamente	Concordo Parcialmente	Nem concordo nem discordo	Discordo Parcialmente	Discordo Amplamente	Discordo Totalmente
Minha **interação** com a linguagem GSE foi **clara e compreensível**							
Interagir com a linguagem GSE **não exige** muito do meu **esforço mental**							
Considero a linguagem GSE **fácil de usar**							
Considero fácil utilizar a linguagem GSE **para fazer o que eu quero que ela faça,** especificar sistemas de forma que todos os stakeholders entendam a especificação							

Comentários sobre benefícios e dificuldades do uso da linguagem GSE:

2. Em relação à sua percepção sobre a **Utilidade da linguagem GSE**, qual o seu grau de concordância em relação às afirmações abaixo (marque com um X):

	Concordo Totalmente	Concordo Amplamente	Concordo Parcialmente	Nem concordo nem discordo	Discordo Parcialmente	Discordo Amplamente	Discordo Totalmente
Usar a linguagem GSE melhorou o meu **desempenho** na especificação de requisitos de sistemas							
Usar a linguagem GSE permitiu aumentar minha **produtividade** na especificação de requisitos de sistemas							
Usar a linguagem GSE aumentou minha **eficácia** na especificação de requisitos de sistemas							
Considero a linguagem GSE **útil** para a especificação de requisitos de sistemas							

LINGUAGEM GSE – Questionário pós-uso

Comentários sobre a utilidade da linguagem GSE:

3. Em relação à sua percepção sobre a **Futuro Uso da linguagem GSE**, qual o seu grau de concordância em relação às afirmações abaixo (marque com um X):

	Concordo Totalmente	Concordo Amplamente	Concordo Parcialmente	Nem concordo nem discordo	Discordo Parcialmente	Discordo Amplamente	Discordo Totalmente
Supondo que eu tenho acesso à linguagem GSE, **eu pretendo usá-la**							
Levando em conta que eu tenho acesso à linguagem GSE, **eu prevejo que eu iria usá-la**							

Comentários sobre o futuro uso da linguagem GSE:

4. Na sua opinião, como a linguagem GSE poderia ser melhorada?

Sugestões de melhoria:

A.2 SOLUÇÃO DO CALENDÁRIO SETORIAL

A seguir encontra-se a solução proposta pelo autor utilizando GSE do problema utilizado durante o experimento.

```
Característica: Inserir evento

Descrição:
    Como um professor
    Ou um técnico administrativo
    Eu quero inserir um evento
    Para que eu possa reservar os recursos necessários

Restrições e qualidades:
    O requisito deve implementar:
    * Uma validação de usuário antes de inserir dados
    O ator não deve ser capaz de:
    * Inserir eventos com dados incompletos

Cenário: Inserção sem erros de um evento
    Dado que eu estou na tela de inserção de evento
    E preenchi o nome do evento
    E o tipo do evento
    E o horário do evento
    E o local do evento
    Quando eu clico em Salvar evento
    Então o sistema deverá salvar o evento
    E mostrar um mensagem avisando que o evento foi inserido com sucesso

Cenário: Não inserção de um evento por erro de preenchimento
    Dado que eu estou na tela de inserção de evento
    E não preenchi todos os campos obrigatórios
    Quando eu clico em Salvar evento
    Então o sistema não deverá salvar o evento
    E o sistema deverá mostrar um mensagem explicando o erro que aconteceu
```

```
Característica: Adicionar usuário

Descrição:
    Como um moderador
    Eu quero adicionar usuários
    Assim eu posso cadastrar usuários com dificuldade em se cadastrar

Restrições e qualidades:
    O ator não deve ser capaz de:
    * Inserir usuários com dados incompletos

Cenário: Criação sem erros de um usuário
    Dado que eu estou na tela de criação de usuário
    E preenchi o nome do usuário
    E o curso do usuário
    E o e-mail do usuário
    E a senha do usuário
    Quando eu clico em Salvar usuário
```

```
    Então o sistema deverá salvar o usuário
    E mostrar um mensagem avisando que o usuário foi criado com sucesso

Cenário: Não criação um usuário por erro de preenchimento
    Dado que eu estou na tela de criação de usuário
    E não preenchi todos os campos obrigatórios
    Quando eu clico em Salvar usuário
    Então o sistema não deverá salvar o usuário
    E o sistema deverá mostrar um mensagem explicando o erro que aconteceu
```

```
Característica: Remover usuários

Descrição:
    Como um moderador
    Eu quero remover usuários
    Para que eu possa remover o acesso de usuários não permitidos


Restrições e qualidades:
    O requisito deve implementar:
    * Uma validação de usuário antes de excluir um usuário

Cenário: Remoção com sucesso do usuário
    Dado que eu estou na tela de informação de usuários
    Quando eu clico em Excluir usuário
    E confirmo a ação
    Então o sistema deverá excluir o usuário
    E o sistema deverá mostrar um mensagem informando que o usuário
     escolhido foi removido
```

```
Característica: Remover evento

Descrição:
    Como um professor
    Ou técnico administrativo
    Eu quero remover um evento que eu adicionei
    Para que eu possa liberar o horário para outro evento caso o que
     eu cadastrei não aconteça

Restrições e qualidades:
    O requisito deve implementar:
    * Uma validação de usuário antes de excluir dados
    O ator não deve ser capaz de:
    * Excluir eventos não criados por ele

Cenário: Remoção com sucesso do evento
    Dado que eu estou na tela de informação de um evento
```

```
    E este evento foi criado por mim
    Quando eu clico em Excluir evento
    E confirmo a ação
    Então o sistema deverá excluir o evento
    E o sistema deverá mostrar um mensagem informando que o evento
     escolhido foi removido

Cenário: Não remoção do evento por não ser o criador do evento
    Dado que eu estou na tela de informação de um evento
    E este evento não foi criado por mim
    Quando eu clico em Excluir evento
    E confirmo a ação
    Então o sistema não deverá excluir o evento
    E o sistema deverá mostrar um mensagem informando o motivo do
     evento não ser removido
```

```
Característica: Moderador remove evento

Descrição:
    Como um moderador
    Eu quero remover um evento
    Para que eu possa cancelar um evento sem depender de um professor

Restrições e qualidades:
    O requisito deve implementar:
    * Uma validação de usuário antes de excluir dados

Cenário: Remoção com sucesso do evento
    Dado que eu estou na tela de informação de um evento
    Quando eu clico em Excluir evento
    E confirmo a ação
    Então o sistema deverá excluir o evento
    E o sistema deverá mostrar um mensagem informando que o evento
     escolhido foi removido
```

```
Característica: Exibir evento

Descrição:
    Como um usuário
    Eu quero visualizar os eventos
    Para que eu possa saber o que acontecerá no meu setor

Restrições e qualidades:
    O requisito deve implementar:
    * Uma forma de filtrar os eventos por:
        > Mês
        > Semana
```

```
        > Curso

Cenário: Visualização dos eventos por filtro
    Dado que eu estou na tela de calendário
    Quando eu escolho um filtro
    Então eu visualizo o calendário utilizando aquele filtro
```

A.3 APRESENTAÇÃO UTILIZADA NO TREINAMENTO

A seguir encontra-se a apresentação de slides utilizada durante o treinamento dos participantes do experimento.

GSE
Gherkin Specification Extension

HENRIQUE VARELLA EHRENFRIED
PPGINF – UFPR
10 de Outubro de 2018

Índice

Apresentação

Quem sou

HENRIQUE VARELLA EHRENFRIED

GRADUADO EM BACHARELADO DE CIÊNCIA DA COMPUTAÇÃO PELA UFPR EM 2017

MESTRANDO EM INFORMÁTICA – PPGINF (PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA)

Trabalho

O TRABALHO CONSISTE EM DESENVOLVER UMA FORMA QUE MELHORE A COMUNICAÇÃO ENTRE DESENVOLVEDORES DE SOFTWARE E CLIENTES

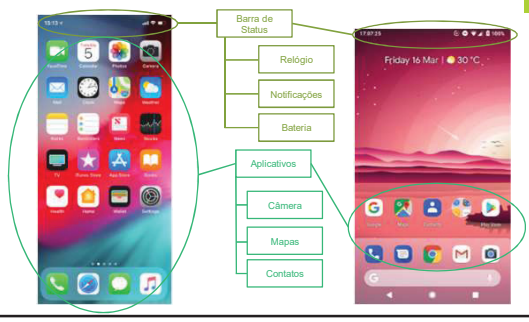
Conceitos

Sistema

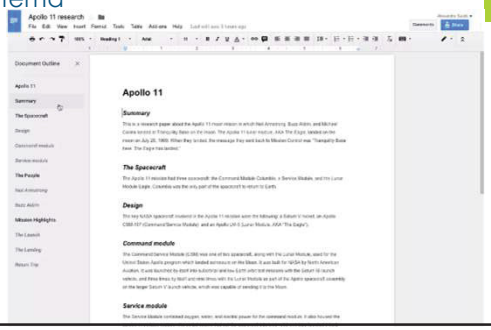
"Um sistema (do grego σύστημα systēma, através do latim systēma), é um conjunto de elementos interdependentes de modo a formar um todo organizado."

- Wikipedia, 26/09/2018

Sistema

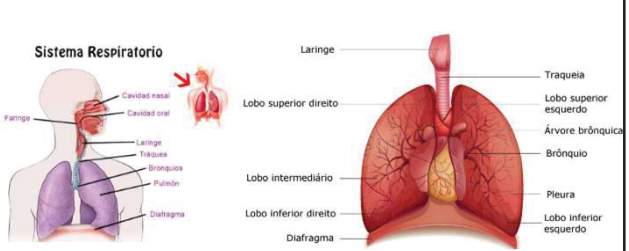


Sistema



Sistema

Sistema



Sistema

Sistema

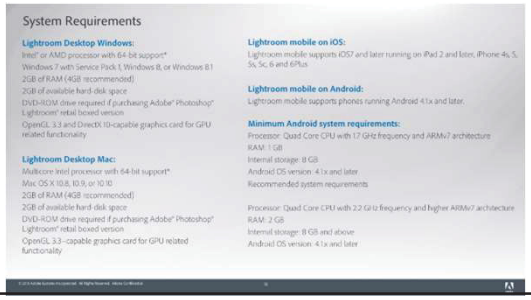
Requisitos - O que é

requisitos

Condições necessárias, geralmente obrigatórias, para se conseguir algo; quesitos: tinha os requisitos para fazer a inscrição.
Exigência básica para se alcançar um propósito: não tenho os requisitos necessários para obter a promoção.

Dicio.com.br

Requisitos - O que é



System Requirements

Lightroom Desktop Windows:
Intel® or AMD processor with 64-bit support*
Windows 7 with Service Pack 1, Windows 8, or Windows 8.1
2GB of RAM (4GB recommended)
2GB of available hard-disk space
DVD-ROM drive required if purchasing Adobe® Photoshop® Lightroom® retail boxed version
OpenGL 3.3 and DirectX 10-capable graphics card for GPU related functionality

Lightroom Desktop Mac:
Multicore Intel processor with 64-bit support*
Mac OS X 10.8, 10.9, or 10.10
2GB of RAM (4GB recommended)
2GB of available hard-disk space
DVD-ROM drive required if purchasing Adobe® Photoshop® Lightroom® retail boxed version
OpenGL 3.3–capable graphics card for GPU related functionality

Lightroom mobile on iOS:
Lightroom mobile supports iOS7 and later running on iPad 2 and later, iPhone 4s, 5, 5s, 5c, 6 and 6Plus

Lightroom mobile on Android:
Lightroom mobile supports phones running Android 4.1x and later.

Minimum Android system requirements:
Processor: Quad Core CPU with 1.7 GHz frequency and ARMv7 architecture
RAM: 1 GB
Internal storage: 8 GB
Android OS version: 4.1x and later

Recommended system requirements

Processor: Quad Core CPU with 2.2 GHz frequency and higher ARMv7 architecture
RAM: 2 GB
Internal storage: 8 GB and above
Android OS version: 4.1x and later

Requisitos - Componentes

Stakeholder
Ator

Funcionalidades

Requisitos - Para que servem

Requisitos - Como especificar

Requisitos - História de Usuário

Título da História	Comprar online
Como um <Usuário/Quem> Eu quero <Ação/O que> Para que <Propósito/Porquê>	Como um consumidor Eu quero comprar online Para que eu consiga comprar o produto que almejo sem sair de casa

Requisitos - História de Usuário

Cenário: Concluir pagamento com sucesso

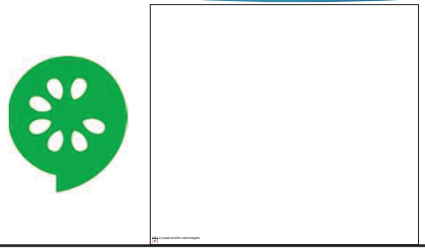
Dado que eu estou na página de pagamento
E que eu preenchi os dados de pagamento corretamente
Quando eu clico em pagar
Então o sistema efetuará o pagamento
E exibirá uma mensagem confirmando o pagamento
E um novo pedido será gerado na minha conta

Dado (estado atual) → Quando (Gatilho de mudança de estado) → Então (novo estado)

I	N	V	E	S	T
• Independent	• Negotiable	• Valuable	• Estimable	• Small	• Testable

GSE

GSE - O que é



Projeto do mestrado

GSE - Para que serve

GSE - Como Funciona

Clientes
- Característica
- Descrição
- Restrições e Qualidades
- Notas

Desenvolvedores
- Grupos
- Planejamento
- Métricas
- Relacionamentos

GSE - Como Funciona

Característica

Característica: Comprar online

Nome da História de Usuário, fornece o que será especificado e uma forma de referenciar o requisito.

GSE - Como Funciona

Descrição

Descrição:

Como um cliente

Eu quero comprar online

Para que eu consiga comprar o produto que almejo sem sair de casa

Define o requisito, isto é, o que o sistema deve fazer. Como segue o padrão de História de Usuário, É definido três itens importantes:

1) Quem
2) O que
3) Para que

GSE - Como Funciona

Restrições e Qualidades

Restrições e qualidades:

O requisito deve implementar:

* Pagamentos via:
  - > Cartão de crédito
  - > Cartão de débito
  - > Boleto

O ator não deve ser capaz de:

* Finalizar a compra sem pagar

Definem detalhes de funcionamento do sistema: Como deve ser feito e os detalhes do que deve existir no sistema.

GSE - Como Funciona

Notas

Notas:

Este requisito foi considerado muito importante pelo cliente

Fornece informações adicionais sobre o requisito. Neste campo pode ser escrito qualquer coisa.

GSE - Como Funciona

Cenários

Cenário: Concluir pagamento com sucesso

Dado que eu estou na página de pagamento
E que eu preenchi os dados de pagamento corretamente
Quando eu clico em pagar
Então o sistema efetuará o pagamento
E exibirá uma mensagem confirmando o pagamento
E um novo pedido será gerado na minha conta

Fornece como o sistema vai funcionar em diversos casos. Logo mais de um cenário pode ser definido

GSE - Como Funciona

Característica: Comprar online

Descrição:

Como um cliente

Eu quero comprar online

Para que eu consiga comprar o produto que almejo sem sair de casa

Restrições e qualidades:

O requisito deve implementar:

* Pagamentos via:
  - > Cartão de crédito
  - > Cartão de débito
  - > Boleto

O ator não deve ser capaz de:

* Finalizar a compra sem pagar

Notas:

Este requisito foi considerado muito importante pelo cliente

Considerações

Considerações – Como utilizar

Para cada requisito antes de seguir adiante

Nomeie característica → Escreva a descrição → Escreva os cenários → Escreva as restrições e qualidades

Para cada requisito antes de seguir adiante

Considerações – Saiba mais

http://dannorth.net/whats-in-a-story/

CHELIMSKY, David. 3 ed. Ensino Superior. The Rspec book, 2010

http://benmabey.com/2008/05/19/imperative-vs-declarative-scenarios-in-user-stories.html

HELM, Rafael; WILDT, Daniel. Ensino Superior. 3 ed. Histórias de Usuário, 2014

http://www.elabs.se/blog/15-you-re-cuking-it-wrong

https://docs.cucumber.io/gherkin/reference/

P. Bourque and R.E. Fairley, eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

Considerações – Dúvidas

Considerações – Exercício

Calendário Setorial

O calendário setorial é um software que permite um setor de uma instituição agendar eventos, reuniões e palestras. Neste sistema, qualquer professor ou técnico administrativo pode incluir um evento.

Este calendário possui um moderador que é capaz de adicionar usuários, remover usuários e remover eventos. Um usuário padrão pode adicionar um evento e remover os eventos que ele mesmo criou. Os usuários são cadastrados pelo moderador. Cada cadastro contém o nome, o curso, o e-mail e a senha do usuário.

Quando um evento é agendado, o usuário que incluiu este agendamento foi requisitado a inserir o nome do evento, seu tipo, horário e local.

A exibição dos eventos agendados poderá ser feita por mês, por semana ou por curso por qualquer pessoa que acesse o calendário setorial.

Obrigado pela atenção

Henrique Varella Ehrenfried
hvehrenfried@inf.ufpr.br
PPGInf – UFPR

A.4 TERMO DE CONSENTIMENTO

A seguir encontra-se o termo de consentimento assinado utilizado no experimento.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Pesquisa: "ANALISANDO A USABILIDADE DA LINGUAGEM DE ESPECIFICAÇÃO DE REQUISITOS GSE"

Prezado Senhor (a),

Como parte de uma pesquisa de mestrado, algumas tecnologias (métodos, técnicas e processos) para avaliação e projeto de modelos das fases iniciais visando a usabilidade da aplicação foram desenvolvidas e têm sido avaliadas experimentalmente. Você está sendo convidado a participar de uma pesquisa que estudará os resultados de utilização dessas tecnologias em uma avaliação/projeto de usabilidade de uma linguagem de especificação de requisitos.
O objetivo da pesquisa é aperfeiçoar estas tecnologias, aumentando sua capacidade de identificação de problemas, contribuindo desta forma para a melhoria da usabilidade da linguagem de especificação de requisitos. Sua participação na pesquisa **não** é obrigatória.

1) Procedimento
A tecnologias serão utilizadas na avaliação de usabilidade da linguagem de especificação de requisitos. Você receberá treinamento sobre as tecnologias e as utilizará para avaliar/projetar a usabilidade das aplicações em questão. Para participar deste estudo solicito a sua especial colaboração em: (1) permitir que os dados resultantes da sua avaliação sejam estudados, (2) informar o tempo gasto na atividade de detecção de problemas ou construção dos modelos de análise de projeto e (3) responder um questionário sobre a utilização das tecnologias. Quando os dados forem coletados, seu nome será removido dos mesmos e não será utilizado em nenhum momento durante a análise ou apresentação dos resultados.

2) Tratamento de possíveis riscos e desconfortos
Serão tomadas todas as providências durante a coleta de dados de forma a garantir a sua privacidade e seu anonimato. Os dados coletados durante o estudo destinam-se estritamente a atividades de pesquisa relacionadas à técnica, não sendo utilizados em qualquer forma de avaliação profissional ou pessoal.

3) Benefícios e Custos
Espera-se que, como resultado deste estudo, você possa aumentar seu conhecimento sobre usabilidade, de maneira a contribuir para o aumento da qualidade de sistemas de software com os quais você trabalhe. Este estudo também contribuirá com resultados importantes para a pesquisa de um modo geral nas áreas de Engenharia de Software e Interface Humano-Computador (IHC).
Você não terá nenhum gasto ou ônus com a sua participação no estudo e também não receberá qualquer espécie de reembolso ou gratificação devido à participação **na pesquisa**.

4) Confidencialidade da Pesquisa
Toda informação coletada neste estudo é confidencial e seu nome e o da sua organização não serão identificados de modo algum, a não ser em caso de autorização explícita para esse fim.

5) Participação
Sua participação neste estudo é muito importante e voluntária. Você tem o direito de não querer participar ou de sair deste estudo a qualquer momento, sem penalidades. Em caso de você decidir se retirar do estudo, favor notificar um pesquisador responsável.
Os pesquisadores responsáveis pelo estudo poderão fornecer qualquer esclarecimento sobre o mesmo, assim como tirar dúvidas, bastando entrar em contato pelos seguintes emails:
Pesquisador: Henrique Varella Ehrenfried - hvehrenfried@inf.ufpr.br – UFPR/PPGINF

6) Declaração de Consentimento
Li ou alguém leu para mim as informações contidas neste documento antes de assinar este termo de consentimento. Declaro que toda a linguagem técnica utilizada na descrição deste estudo de pesquisa foi explicada satisfatoriamente e que recebi respostas para todas as minhas dúvidas. Confirmo também que recebi uma cópia deste Termo de Consentimento Livre e Esclarecido. Compreendo que sou livre para me retirar do estudo em qualquer momento, sem qualquer penalidade. Declaro ter mais de 18 anos e dou meu consentimento de livre e espontânea vontade para participar deste estudo.

Local: Curitiba - PR **Data:** ____-____-____

Participante	Pesquisador
Nome:_____ Assinatura:_____	Nome: Henrique Varella Ehrenfried Assinatura:

Obrigado pela sua colaboração!

A.5 GRAMÁTICA

A seguir encontra-se a gramática da linguagem GSE criada utilizando a ferramenta textX.

```
//noskipws: Do not skip whitespaces. by default textX skips whitespaces

GSE_MODEL[noskipws]:
   'Feature:' Feature=WORD_WITH_SPACE NEWLINE
   Description=DESCRIPTION?
   (
      Group=GROUP?
      Constraints=CONSTRAINTS?
      Relationship=RELATIONSHIP?
      Planning=PLANNING?
      Metrics=METRICS?
   )#
   Notes=NOTES?
   Cucumber=START_CUCUMBER?
;
DESCRIPTION[noskipws]:
   'Description:' NEWLINE
   'As' Stakeholder=TEXT NEWLINE
   ('Or' Stakeholder+=TEXT NEWLINE)*
   'I want to' Action=TEXT NEWLINE
   'So that I' Motivation=TEXT NEWLINE
;
GROUP[noskipws]:
   'Group:' NEWLINE
   Groups+=GROUP_TAG+ NEWLINE
;
CONSTRAINTS[noskipws]:
   'Constraints and Qualities:' NEWLINE
   ('The requirement should implement:' NEWLINE Qualities+=ITEM+ NEWLINE)?
   ('The actor should not be able to:' NEWLINE Constraints+=ITEM+ NEWLINE)?
;
RELATIONSHIP[noskipws]:
   'Relationship:' NEWLINE
   'This requirement is:' NEWLINE
   (
      ('* Derived from requirement(s)' SPACE
       Derived=REQUIREMENT_LIST NEWLINE)?

      ('* Contained in requirement(s)' SPACE
       Contains=REQUIREMENT_LIST NEWLINE)?

      ('* A copy of the requirement(s)' SPACE
       Copy=REQUIREMENT_LIST NEWLINE)?

      ('* A refinament of the requirement(s)' SPACE
       Refinament=REQUIREMENT_LIST NEWLINE)?

      ('* Dependent on requirement(s)' SPACE
       Dependent=REQUIREMENT_LIST NEWLINE)?
   )#
;
PLANNING[noskipws]:
   'Planning:' NEWLINE
```

```
54    'This requirement will be implemented' SPACE Order=ORDINAL_NUM SPACE
55    'and will take' SPACE Time_needed=TIME SPACE
56    'to be implemented by' SPACE Num_of_developers=/\d+/
57    SPACE 'developer(s).' NEWLINE
58    'The developer(s) responsible for implementing this requirement is/are'
59    SPACE Developers+=WORD_WITH_SPACE[','] SPACE '.' NEWLINE
60    'The sprint that will implement this feature is:' SPACE
61    Sprint=WORD_WITH_SPACE SPACE '.' NEWLINE
62 ;
63 METRICS[noskipws]:
64    'Metrics:' NEWLINE
65    'The metrics used to evaluate this requirement are:' NEWLINE
66    (Metric_item+=METRIC_ITEM)+ NEWLINE
67 ;
68 NOTES[noskipws]:
69    'Notes:' NEWLINE
70    (Note+=TEXT NEWLINE)+
71 ;
72
73 START_CUCUMBER[noskipws]:
74    'Scenario:'?
75    'Background:'?
76    'Scenario Outline'?
77    'Example:'?
78    'Examples:'?
79    (ANYTHING* NEWLINE*)*
80 ;
81 ANYTHING[noskipws]:
82    /(.*)*/
83 ;
84 GROUP_TAG[noskipws]:
85    // (Tag+=WORD* NEWLINE | ('[' Tag+=WORD_WITH_SPACE ']')* NEWLINE)*
86    Tag+=WORD_WITH_SPACE[',']
87 ;
88 METRIC_ITEM[noskipws]:
89    '*' SPACE Metric+=WORD_WITH_SPACE SPACE ';' NEWLINE
90    ('The expected value is at least' SPACE
91     Expected_Value+=WORD_WITH_SPACE NEWLINE)?
92 ;
93 TIME[noskipws]:
94    /\d*/ SPACE TIME_UNIT
95 ;
96 TIME_UNIT[noskipws]:
97    ('second' | 'seconds' | 'minute' | 'minutes' |
98     'hours' | 'hours' | 'day' | 'days' | 'week' |
99     'weeks' | 'month' | 'months' | 'year' | 'years')
100 ;
101 ORDINAL_NUM[noskipws]:
102    /\d*/('st' | 'nd' | 'rd' | 'th')
103 ;
104 REQUIREMENT_LIST[noskipws]:
105    Requirement+=WORD_WITH_SPACE[',']
106 ;
107 WORD_WITH_SPACE[noskipws]:
108    /[\w*\'* ]*/
109 ;
110 ITEM[noskipws]:
111    '*' SPACE Item+=TEXT NEWLINE Subitem+=SUBITEM?
```

```
112 ;
113 SUBITEM[noskipws]:
114    '>' SPACE Subitem+=TEXT NEWLINE SUBITEM?
115 ;
116 TEXT[noskipws]:
117    /[^\r\n\[\]]*/
118 ;
119 WORD[noskipws]:
120    /(\w)*/
121 ;
122 NEWLINE[noskipws]:
123    /[\r\n\t ]*/
124 ;
125 SPACE[noskipws]:
126    /[\t ]*/
127 ;
```