UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

YURI CAVALCANTE AMANCIO

SIGFINDER - REPOSITÓRIO ONLINE DE GRÁFICOS DE INTERDEPENDÊNCIA DE SOFTGOALS

QUIXADÁ

2018

YURI CAVALCANTE AMANCIO

SIGFINDER - REPOSITÓRIO ONLINE DE GRÁFICOS DE INTERDEPENDÊNCIA DE SOFTGOALS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientadora: Profa. Ma. Rainara Maia Carvalho

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A499s Amancio, Yuri Cavalcante.
SIGFinder - Repositório online de gráficos de interdependência de softgoals / Yuri Cavalcante Amancio. – 2018.
65 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2018.
Orientação: Profa. Ma. Rainara Maia Carvalho.

1. Requisitos não-fucionais (Engenharia de Sistemas). 2. Requisitos não-funcionais - Catálogos. 3. Bibliotecas Digitais. I. Título.

CDD 005.1

YURI CAVALCANTE AMANCIO

SIGFINDER - REPOSITÓRIO ONLINE DE GRÁFICOS DE INTERDEPENDÊNCIA DE SOFTGOALS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: __/__/__

BANCA EXAMINADORA

Profa. Ma. Rainara Maia Carvalho (Orientadora)
Universidade Federal do Ceará (UFC)

Profa. Dra. Rossana Maria de Castro Andrade
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcio Espíndola Freire Maia
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

A minha família, por sempre ter me apoiado e me ajudado, foram anos de muitas lutas, mas com a graça de Deus estou aqui hoje. Em especial à minha mãe, obrigado por ter se esforçado tanto, nós estamos nos formando juntos.

Aos meus amigos e irmãos da igreja a qual faço parte, MANT. Vocês me mostraram muitas vezes o amor de Jesus por mim através de suas vidas. Pastores Vanderley e Ilda Sousa, vocês foram mais do que pastores, e sim pais para mim. Obrigado por cada conselho e orientação, sintam-se parte dessa conquista também.

Aos meus amigos que fiz aqui em Quixadá, muitos de vocês foram como pais e mães para mim. Wellington Rocha, meu pastor aqui nesta cidade, obrigado por todo o apoio e por deixar eu ver Jesus através do senhor. Breno e Wládia, obrigado por todo o apoio e todas as mudanças, vocês me ajudaram muito.

A minha orientadora Rainara Maia Carvalho, muito obrigado pela direção dada nesses meses de trabalho, foi uma honra ser seu orientando.

A Universidade Federal do Ceará (UFC) e a todos os professores do campus Quixadá, que me auxiliaram nessa caminhada, foi um prazer fazer parte desse campus.

Por último, mas o mais importante, ao Espírito Santo, meu melhor amigo. O Senhor nunca me deixou, e sempre esteve comigo, nos momentos difíceis e nos momentos fáceis, obrigado por tudo.

“mas, como está escrito: Nem olhos viram, nem ouvidos ouviram, nem jamais penetrou em coração humano o que Deus tem preparado para aqueles que o amam”

(1 Coríntios 2:9 - Bíblia Sagrada)

RESUMO

Requisitos não-funcionais têm impacto direto na qualidade de um software desenvolvido. Por assim ser, os desenvolvedores necessitam esforçar-se para alcançar os RNFs que são precisos para entregar o software requerido com qualidade. RNFs são globais em sua natureza. Por isso, enquanto os desenvolvedores esforçam-se para atingir os RNFs da aplicação, podem ocorrer conflitos e sinergias entre os mesmos. Para auxiliá-los nos *trade-offs* existentes, a equipe responsável por tal atividade pode fazer uso de conhecimento armazenado por terceiros, poupando assim tempo, custo e esforço. Esses conhecimentos armazenados geralmente estão representados em forma gráfica, na forma de Softgoal Interdependency Graphs (SIGs). Entretanto, estes catálogos estão espalhados em várias bases de dados e livros, cuja busca por eles é dispendiosa. Para solucionar o problema abordado, o presente trabalho apresenta uma aplicação web, denominada SIGFinder, onde estes catálogos estão categorizados e disponibilizados para os desenvolvedores, poupando assim o tempo de busca que seria necessário para encontrá-los. Por meio de um teste de usabilidade, a aplicação foi avaliada por usuários que fizeram um conjunto de tarefas propostas. Os resultados deste teste mostraram um bom nível de satisfação dos participantes em relação a aplicação, bem como um bom nível de facilidade ao executarem as atividades propostas. Conclui-se que o repositório é capaz de armazenar e disponibilizar, e mostrar aos usuários os catálogos de RNFs. Além disso, também foi realizada uma avaliação da qualidade interna do código, a qual indica que a aplicação possui poucas vulnerabilidades.

Palavras-chave: Requisitos não-funcionais. Catálogos de Requisitos não-funcionais. Repositórios Digitais

ABSTRACT

Non-functional requirements have a direct impact on the quality of developed software. In so doing, developers need to strive to reach the RNFs that are needed to deliver the required software with quality. RNFs are global in nature, so as developers strive to reach the application's RNFs, conflicts and synergies can occur between them. To assist them in existing trade-offs, the team responsible for such activity can make use of knowledge stored by third parties, thus saving time, cost and effort. This stored knowledge is usually represented graphically, in the form of Softgoal Interdependency Graphs (SIGs). However, these catalogs are scattered in various databases and books, the search for which is expensive. In order to solve the problem addressed, the present work presents a tool, called SIGFinder, where these catalogs are categorized and made available to the developers, thus saving the search time that would be necessary to find them. Through a usability test, the application was evaluated by users who did a set of proposed tasks. The results of this test showed a good level of participants' satisfaction with the application, as well as a good level of ease in carrying out the proposed activities. It is concluded that the repository is able to store and make available, and to show users the catalogs of RNFs. In addition, an internal quality assessment of the code was also performed, which indicates that the application has few vulnerabilities.

Keywords: Non-Functional Requirements. Softgoal Interdependency Graphs. Repositories

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES E SIGLAS

TI	Tecnologia da Informação
SCV	Sistema de Controle de Versão
RNF	Requisito Não-Funcional
SIG	Softgoal Interdependecy graphs
UML	Unified Modeling Language
CASE	Computer-Aided Software Engineering
API	Application Programming Interface

SUMÁRIO

1 INTRODUÇÃO

Nos dias atuais, é cada vez mais necessário que os profissionais responsáveis por elicitar e compreender requisitos de software, estejam atentos aos requisitos não-funcionais do software (RNFs), pois eles podem distinguir entre um produto que meramente faz o que é proposto de um produto que vai além e satisfaz o usuário (WIEGERS K. ; BEATTY, 2013). RNFs são aqueles relacionados às características de qualidade que um software pode possuir, tais como Segurança, Usabilidade e Desempenho (WIEGERS K. ; BEATTY, 2013).

Esses requisitos podem estar frequentemente interagindo entre si. Isso significa que enquanto há um esforço para atingir certo requisito, implicações positivas ou negativas em outros requisitos podem ocorrer (CHUNG *et al.*, 2000). Essas interações são definidas como conflito e sinergia (CHUNG *et al.*, 2000).

Tais interações entre RNFs são frequentes em softwares (CYSNEIROS, 2001). Um conflito acontece quando apoiar um RNF pode impactar negativamente outro RNF. Por exemplo, um conflito comum na literatura é o de Segurança e Desempenho. Caso o desenvolvedor decida apoiar um, existe uma grande chance de impactar de forma negativa o outro. Por sua vez, harmonias entre RNFs ocorrem quando apoiar um RNF pode impactar de forma positiva outro RNF. Por exemplo, a escolha de Segurança impacta positivamente em Confiabilidade.

É importante identificar o quanto antes esses conflitos e sinergias em tempo de desenvolvimento do software para que a equipe não se comprometa com RNFs conflitantes. Por essa razão, catálogos são disponibilizados para apoiar os desenvolvedores a identificarem interações entre os requisitos não-funcionais e ajudam também a alcançarem um *trade-off* entre diferentes RNFs.

Catálogos de requisitos não-funcionais são uma forma de armazenar o conhecimento sobre a interação entre RNFs (CHUNG *et al.*, 2000). Catálogos podem ser utilizados para auxiliar desenvolvedores enquanto o mesmo estiver no processo de análise de *trade-off* do sistema que está trabalhando no momento ou até mesmo antes desta análise, para se obter conhecimento prévio de possíveis soluções para o seu problema atual.

Uma notação existente e comumente utilizada na literatura para catalogar conhecimento de RNFs é a notação SIG (do inglês *Softgoal Interdependency Graph*) que foi proposto pelo *NFR Framework*. Tal notação consegue apresentar um nível mais específico de como interações podem ocorrer, apoiando de forma mais concreta os desenvolvedores.

Embora existam diversos catálogos em formato de SIGs disponíveis (CARVALHO

et al., 2018), (MAIA *et al.*, 2009), (CHUNG *et al.*, 2000), nem sempre é tão fácil encontrá-los. Muitos estão espalhados em diversas bibliotecas *online* de trabalhos científicos, tornando difícil e dispendiosa a busca por eles. Embora existam algumas ferramentas na literatura que apoiem esses catálogos, como a *NFR-Assistant* (TRAN QUAN.; CHUNG, 1999) e a *NDR-Tool* (SUPAKKUL; CHUNG, 2012), elas possuem algumas limitações. Enquanto uma está disponível apenas para a plataforma desktop e não possui catálogos armazenados *desktop* (TRAN QUAN.; CHUNG, 1999), a outra não se encontra disponível para download (SUPAKKUL; CHUNG, 2012).

Dessa forma, esse trabalho objetiva construir um repositório online de catálogos SIGs, chamado SIGFinder, para consultoria e pesquisa pelos profissionais da área de requisitos e projetistas. O repositório pode conter diversos catálogos divididos por domínio. A divisão faz-se necessária pois em um domínio dois requisitos podem gerar um conflito e em outro domínio os mesmos requisitos podem gerar uma harmonia. Nesse repositório proposto, o usuário poderá não apenas visualizar e buscar os catálogos que já estarão disponíveis previamente na base de dados, mas também adicionar catálogos que foram criados por ele.

Por meio desta interação, o repositório pode estar constantemente sendo atualizado pelo próprios usuários que terão seus catálogos disponibilizados no repositório. Esta constante atualização pelos usuários poderá engajá-los ainda mais no uso do repositório.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta os conceitos teóricos envolvidos no contexto do trabalho. A Seção 3 mostra a descrição dos procedimentos metodológicos adotados na realização deste trabalho. A Seção 4 apresenta o repositório de catálogos de RNFs em formato de SIGs proposto nesse trabalho, chamado de SIGFinder. A Seção 5 apresenta a avaliação realizada nesse repositório. A Seção 6 apresenta os estudos relacionados correspondentes a ideia proposta por este trabalho, a Seção 7 apresenta as conclusões desse trabalho. Por fim, foram adicionados 5 apêndices a este trabalho, que são documentos criados durante a execução do repositório, todos foram utilizados no teste de usabilidade, exceto o E, que foi o diagrama de classe criado durante a fase de modelagem e também de desenvolvimento do repositório. Os apêndices são: Apêndice A (Termo de consentimento), Apêndice B (Questionário pré-teste), Apêndice C (Cenário das Tarefas), Apêndice D (Questionário pós-teste) e Apêndice E (Diagrama de Classe).

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentadas as definições e descrições dos principais conceitos que são necessários para esse projeto.

2.1 Repositórios Digitais

Segundo (HEERY; ANDERSON, 2005), um repositório é uma coleção de objetos digitais. Além disso, repositórios digitais diferem de outras coleções digitais como diretórios, catálogos e banco de dados por várias razões, algumas delas são: o conteúdo é depositado, seja por seu criador ou por terceiros; o repositório deve fornecer um conjunto mínimo de serviços básicos, como cadastrar, obter e recuperar dados, além de controlar acesso; o repositório deve ser sustentável e confiável, além de ser bem gerenciado (HEERY; ANDERSON, 2005).

Ainda que nesta primeira versão algumas funcionalidades que são intrínsecas à definição de repositório (*e.g.*, controle de acesso) possam não ser contempladas, por meio das características citadas acima, a ferramenta proposta por esse trabalho pode ser classificada na categoria de repositório digital.

Repositórios digitais podem ser utilizados por vários tipos de usuários alvo. No trabalho atual, SIGFinder pode ser acessada tanto por profissionais da área de requisitos e desenvolvedores, quanto por pesquisadores que desejam encontrar os catálogos disponibilizadas na ferramenta quanto por estudantes que desejam aprimorar o conhecimento sobre *RNFs* e mais precisamente *SIG*, contribuindo assim com as diferentes áreas relacionadas ao uso de *SIGs*.

2.2 Definições de RNFs

Existe uma relação direta entre a satisfação dos clientes com a correta implementação desse tipo de requisito de software (WIEGERS K. ; BEATTY, 2013). Por essa razão, *RNFs* têm suma importância no sucesso de sistemas de *software* (MAIRIZA; ZOWGHI, 2010).

Segundo a definição de (SOMMERVILLE, 2011), *RNFs* são requisitos que não estão diretamente relacionados com os serviços específicos disponibilizados pelo sistema a seus usuários. Entretanto, (WIEGERS K. ; BEATTY, 2013) conceitua *RNFs* como uma propriedade ou característica que um sistema deve exibir ou uma restrição que deve ser respeitada. Neste trabalho, considera-se que requisitos não-funcionais são uma junção das duas definições abordadas acima, *RNFs* apesar de não estarem ligados diretamente a funções do sistema, eles que ditam

como o software irá se comportar por meio de suas características.

RNFs podem ser classificados em internos e externos (WIEGERS K. ; BEATTY, 2013). Requisitos internos estão ligados a técnicas pertinentes ao desenvolvimento do código fonte do *software* ou de sua arquitetura que indiretamente irão afetar o usuário do sistema. Alguns exemplos de *RNFs* internos são:

- Eficiência - Quão eficiente o sistema utiliza os recursos do computador.
- Portabilidade - Quão facilmente o sistema pode ser feito para trabalhar em outros ambientes operacionais.
- Manutenibilidade - Quão fácil é manter, mudar, aprimorar e reestruturar o sistema.

.

Se um software não foi desenvolvido utilizando boas práticas de desenvolvimento, por exemplo, pode afetar na manutenibilidade do código e dificultar a adição de novas funcionalidades.

Por outro lado, *RNFs* externos são características que são observadas enquanto o software está executando (WIEGERS K. ; BEATTY, 2013). Exemplos de RNFs externos são listados abaixo:

- Disponibilidade - Até que ponto os requisitos do sistema estão disponíveis quando e onde eles são necessários.
- Performance - Quão rápida e previsivelmente o sistema responde a entrada de usuário e outros eventos.
- Robustez - Quão bem o sistema responde a condições de operações não esperadas [1].

.

2.3 Interações entre *RNFs*

Dada a definição de requisitos não-funcionais, faz-se necessário entender que eles não estão contidos em um *software* isolados uns dos outros, ou seja, existe interação entre eles. *RNFs* são globais em sua natureza por isso algumas decisões relacionadas ao *design* do produto podem ter efeito em outra parte do sistema (CHUNG *et al.*, 2000). Por essa razão no desenvolvimento de um *software* é preciso elicitar os *RNFs* tendo em vista sempre a viabilidade e o objetivo de negócio do produto que irá ser gerado, decidindo quais requisitos serão implementados, e quais serão priorizados. Este processo é comumente chamado *trade-off*.

[1] Todas as definições de tipos de RNFs foram retiradas da seguinte fonte: (WIEGERS K. ; BEATTY, 2013)

Em qualquer sistema de *software* que haja mais de um *RNF*, pode ocorrer o *trade-off* entre eles. Por exemplo: um sistema onde o cliente deseja que seu software seja rápido e ao mesmo tempo seguro pode resultar em um *trade-off*, em outras palavras ele precisará decidir qual será priorizado ou até mesmo desistir da implementação de um, em via do outro. Existem dois tipos de interações entre *RNFs*: conflitos e sinergias.

Conflitos entre *RNFs* são frequentes em muitos sistemas, eles ocorrem entre dois *RNFs* quando é impossível ou difícil conseguir atingí-los ao mesmo tempo (BERANDER *et al.*, 2005). A quantidade de conflitos pode estar associada ao tamanho do software e a quantidade de *stakeholders* envolvidos (BERANDER *et al.*, 2005), pois quanto mais pessoas envolvidas maior a chance de existirem conflitos de interesses.

Muitas vezes durante a elicitação dos requisitos, os conflitos são percebidos pelos analistas de requisitos e já são expostos para que uma decisão seja tomada pelo cliente. Em outros casos, os conflitos só aparecem durante o *design* da arquitetura do sistema, ou em piores casos, durante o desenvolvimento do software. É de extrema importância que os conflitos sejam reportados ao cliente tão logo são descobertos, pois a falta de priorização de alguma característica do sistema pode gerar insatisfação do cliente.

Por exemplo, um cliente almeja um software para seu comércio eletrônico e deseja que o *e-commerce* seja rápido e ao mesmo tempo robusto para evitar que o seu negócio seja defraudado por terceiros. Neste caso, há um conflito entre os *RNFs* citados, pois a robustez de um sistema implica em ter mais camadas de segurança afetando assim o desempenho da aplicação.

Já sinergias acontecem quando um requisito impacta de forma positiva outro requisito, em outras palavras, *RNFs* podem auxiliar outros *RNFs* (CHUNG *et al.*, 2000). Quanto mais sinergias existirem em um software, melhor será, pois os desenvolvedores não necessitarão priorizar ou fazer uma análise de *trade-offs*. Utilizando o exemplo acima, o cliente pode também solicitar que haja integridade nos dados que irão trafegar pelo sistema. Assim sendo, esse *RNF* impacta positivamente a robustez, gerando assim uma sinergia.

2.4 Catálogos de requisitos não-funcionais

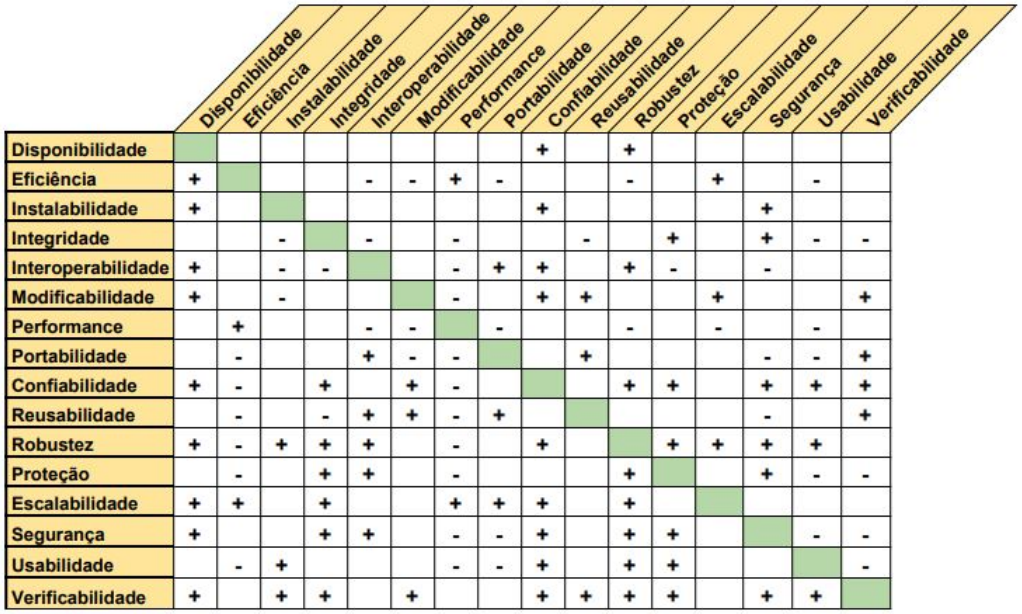
Catálogos de requisitos não-funcionais são uma forma de armazenar o conhecimento sobre os conflitos e sinergias entre *RNFs* (CHUNG *et al.*, 2000). Por meio de catálogos podem ser inferidas várias informações que auxiliam os engenheiros de requisitos a perceberem os

trade-offs existentes bem como analisar qual a melhor solução para os conflitos e sinergias que foram identificados. Neste trabalho os catálogos são categorizados da seguinte forma: catálogos em forma de matriz e em forma gráfica.

2.4.1 Catálogos em Matriz

Este tipo de catálogo contém informações entre interações de requisitos não-funcionais do *software*, em outras palavras são catálogos que disponibilizam as interações existentes entre os *RNFs*. A Figura 1 apresenta um catálogo em forma de matriz com relacionamentos entre tipos de *RNFs*.

Figura 1 – Relacionamento entre tipos de *RNFs*



	Disponibilidade	Eficiência	Instalabilidade	Integridade	Interoperabilidade	Modificabilidade	Performance	Portabilidade	Confiabilidade	Reusabilidade	Robustez	Proteção	Escalabilidade	Segurança	Usabilidade	Verificabilidade
Disponibilidade								+		+						
Eficiência	+				-	-	+	-			-		+		-	
Instalabilidade	+								+					+		
Integridade			-		-		-			-		+		+	-	-
Interoperabilidade	+		-	-			-	+	+		+	-		-		
Modificabilidade	+		-				-		+	+			+			+
Performance		+			-	-		-			-		-		-	
Portabilidade		-			+	-	-			+				-	-	+
Confiabilidade	+	-		+		+	-				+	+		+	+	+
Reusabilidade		-		-	+	+	-	+						-		+
Robustez	+	-	+	+	+		-		+			+	+	+	+	
Proteção		-		+	+		-				+			+	-	-
Escalabilidade	+	+		+			+	+	+		+					
Segurança	+			+	+		-	-	+		+	+			-	-
Usabilidade		-	+				-	-	+		+	+				-
Verificabilidade	+		+	+		+			+	+	+	+		+	+	

Fonte: (WIEGERS K. ; BEATTY, 2013) - adaptado pelo autor

No quadro, sinais de menos" significam que conforme o atributo da linha for aumentando no projeto, o atributo da coluna é afetado negativamente. Uma célula vazia expressa que o atributo da linha tem pouco efeito sobre o que está na coluna. Já o sinal de mais" expressa que o atributo da linha impacta positivamente o da coluna.

2.4.2 Catálogos em Gráficos

Durante o processo de elicitar os *RNFs* no projeto, os profissionais da área de requisitos necessitam tomar uma série de decisões para embasar as escolhas feitas. Uma forma de realizar isso é criando catálogos gráficos. Por meio destes é possível não apenas visualizar quais *RNFs* são prioritários no *software*, mas entender o motivo pelo qual determinado *RNF* foi escolhido em detrimento de outro.

Uma notação existente e comumente utilizada para catalogar conhecimento de *RNFs* é a notação SIG (do inglês *Softgoal Interdependency Graph*) que foi proposto pelo *NFR Framework*, ambos conceitos estão melhor descritos nas subseções abaixo.

2.4.2.1 NFR Framework

Como definido em (CHUNG *et al.*, 2000), o *NFR Framework* oferece uma estrutura para representar e registrar o processo de projeto e raciocínio em grafos, chamados grafos de interdependência entre *softgoals* (SIGs, este conceito será abordado mais detalhadamente na subseção abaixo). Em concordância, (TRAN; CHUNG, 1998) afirma que o *NFR Framework* provê uma simples linguagem para aplicadamente mostrar os *RNFs*. Sua linguagem de fácil entendimento oferece a possibilidade do uso dos grafos que são construídos para documentação de requisitos não-funcionais. Dessa forma, este *framework* lida com os requisitos como comportamentos do sistema e não com funcionalidades (CHUNG *et al.*, 2000).

O objetivo do *NFR Framework* é simplificar e ao mesmo tempo demonstrar os RNFs de uma forma que o desenvolvedor consiga compreender melhor as relações entre os requisitos não-funcionais (LAPOUCHNIAN, 2005).

O *NFR Framework* tem como base o conceito de *softgoals* que representa metas que não têm uma definição clara e/ou critérios que mostram se esta é, por sua vez, satisfeita ou não (CHUNG *et al.*, 2000). Por meio dos *softgoals*, o desenvolvedor pode utilizar RNFs como metas e refiná-las em outros *softgoals* e assim sucessivamente para que ao final do processo de criação do grafo, o desenvolvedor consiga identificar o que é necessário para conseguir atingir os *RNFs* que foram estabelecidos.

Nesta abordagem o profissional tem a possibilidade de fazer o refinamento entre *RNFs* adicionando operacionalizações ao grafo. Operacionalização é uma estratégia real que o desenvolvedor pode utilizar para satisfazer um determinado *softgoal*.

2.4.2.2 Softgoal Interdependency Graph (SIG)

Um *SIG* é um grafo de *softgoals* interconectados um ao outro onde cada um representa um *RNF* para o sistema que está sendo desenvolvido (YRJÖNEN; MERILINNA, 2009). SIGs são uma notação bem conhecida no processo de catalogar *RNFs* e seus tipos (como atributos de qualidade) (CARVALHO *et al.*, 2018). Neste gráfico, cada requisito vai sendo decomposto em alvos menores, onde cada alvo tem um impacto positivo ou negativo sobre o alvo do qual ele descende.

Existem algumas notações que são utilizadas neste tipo de gráfico que simbolizam o tipo de contribuição entre os *sofgoals*, essas estão descritas na Tabela 1:

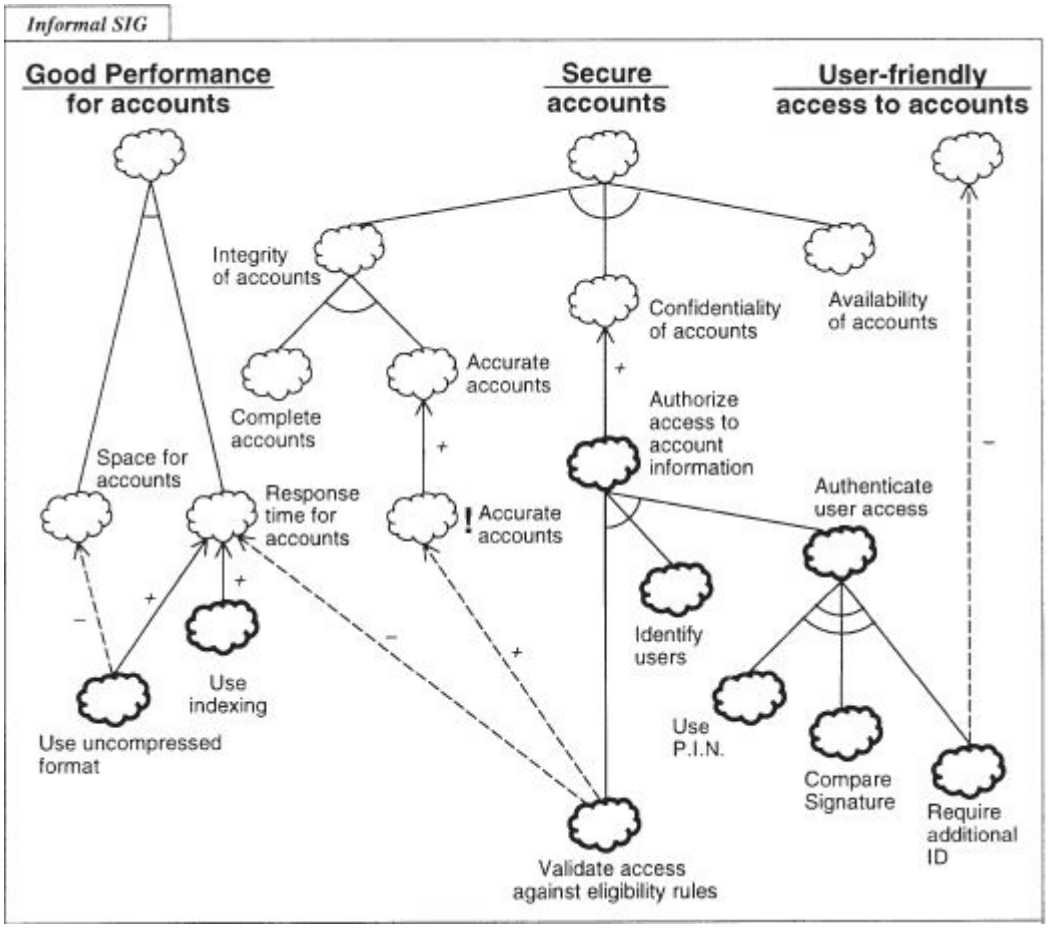
Tabela 1 – Tabela com contribuições SIGs.

Símbolo	Contribuição	Descrição
++	MAKE	O *Softgoal* filho é fortemente propagado ao pai.
+	HELP	O *Softgoal* filho é de alguma forma propagado ao pai.
=	EQUAL	Os dois *softgoals* compartilham do mesmo rótulo.
-	HURT	A negação do *softgoal* filho é de alguma forma propagada ao pai.
–	BREAK	*Softgoal* filho é fortemente negado e propagado ao pai.
?	UNKNOWN	Interdependência desconhecida, o *softgoal* filho não afeta o pai.

Fonte: (YRJÖNEN; MERILINNA, 2009), adaptada pelo autor

A Figura 2 apresenta um exemplo de um grafo *SIG* construído para atender os *RNFs*: desempenho e segurança para contas e acesso amigável ao usuário, também para contas:

Figura 2 – Exemplo de SIG



Fonte: Chung *et al.* (2000)

No exemplo acima, o desenvolvedor foi decompondo os *softgoals* principais em outros *softgoals*, até o nível que foi necessário para cada RNF. Note que alguns dos *softgoals* contidos neste grafo são operacionalizações adicionadas pelo desenvolvedor, por exemplo: *Use P.I.N*". Note ainda que existem *softgoals* que impactam outros, por exemplo *Use Indexing*" afeta positivamente, ou seja, essa operacionalização auxilia na conquista de *Response time for accounts*".

Em um grafo *SIG* também é possível ao desenvolvedor sinalizar que determinado *softgoal* deve ter prioridade no grafo (CHUNG *et al.*, 2000). Uma priorização pode ser vista no *softgoal Accurate accounts*".

3 TRABALHOS RELACIONADOS

Nesta seção serão abordados os trabalhos relacionados que mais se assemelham a solução proposta por este projeto. Todos eles possuem características em comum com a solução apresentada por este trabalho.

3.0.1 NFR-Assistant

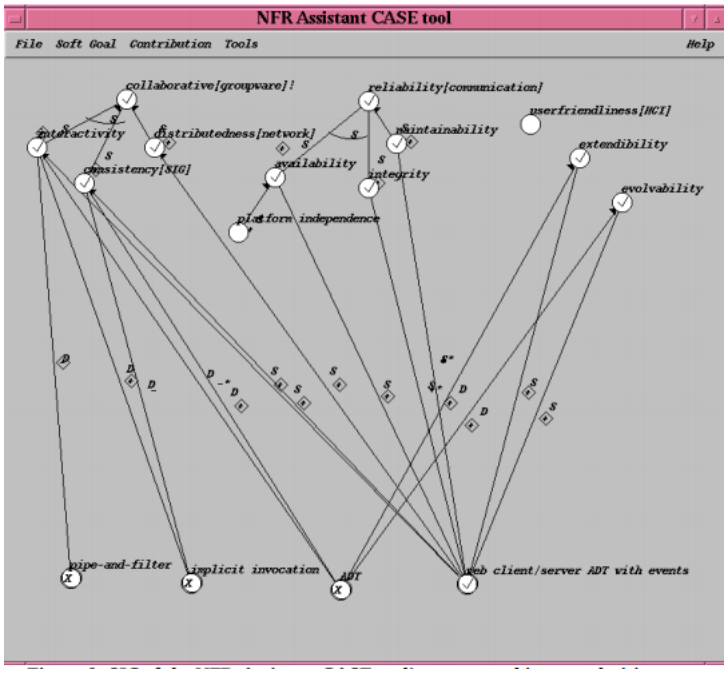
NFR-Assistant é uma ferramenta CASE que incorpora a semântica e a sintaxe do *NFR Framework* em uma aplicação prática para auxiliar os desevolvedores de software na análise dos requisitos não-funcionais (TRAN; CHUNG, 1998).Uma de suas características é que essa ferramenta permite a múltiplos usuários, a criação e alteração de um mesmo SIG, possibilitando assim que vários stakeholders façam parte da modelagem dos requisitos não-funcionais do software.

Embora *NFR-Assistant* não seja um repositório de catálogos, nela os usuários têm a possibilidade de criar vários catálogos para diversos sistemas que precisarem ser desenvolvidos. Por fazer uso do *framework* NFR e também uso de catálogos SIGs, esta ferramenta se assemelha à solução proposta por este trabalho, porém não fornece aos usuários catálogos de RNFs que possam otimizar o processo de análise dos RNFs e nem mesmo está disponível para download ou acessível *online*.

No final desta seção serão apresentadas as semelhanças e diferenças entre os trabalhos relacionados e o repositório proposto por este trabalho em forma de tabela.

A seguir a Figura 4 apresenta um exemplo de um SIG criado com a *NFR-Assitant*:

Figura 3 – SIG criado na ferramenta NFR-Assistant em um processo de decisão arquitetural de um sistema



Fonte: (TRAN QUAN.; CHUNG, 1999)

3.0.2 NDR Tool

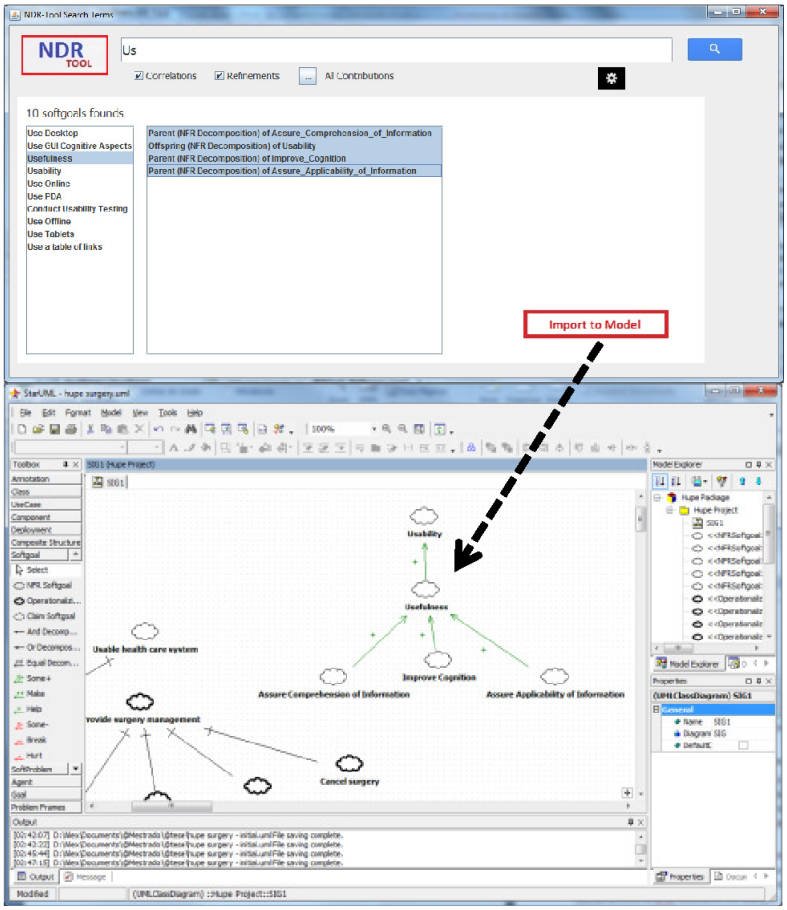
NDR Tool é uma extensão que pode ser utilizada na ferramenta Star-UML/RE-Tool Designer (SUPAKKUL; CHUNG, 2012), que permite ao usuário montar seu gráfico *SIG* a partir de catálogos já disponíveis na própria ferramenta. O engenheiro de software utiliza a NDR-Tool, para realizar buscas nos catálogos e integrar os requisitos encontrados à modelagem atual e, assim, incluir novos *NFRs* à modelagem, tendo em mente as implicações que estas podem causar no *software* (ARAÚJO *et al.*, 2014).

A ferramenta *NDR-Tool* possui catálogos de *RNFs* disponíveis para consulta categorizados por tipos (segurança e usabilidade, por exemplo) assemelhando-se assim ao que este trabalho propõe. Porém esta ferramenta não disponibiliza-os de forma online nem categoriza-os por domínio (sistema bancário e software médico, por exemplo).

A solução que é proposta neste trabalho não é apenas possuir um repositório de catálogos, mas categorizá-los por domínio, permitir que os desenvolvedores adicionem catálogos ao repositório e viabilizar o acesso ao máximo de pessoas por meio da Internet contribuindo assim para que os profissionais da área de requisitos despendam menos tempo na análise dos *RNFs* de um sistema de *software*.

A Figura 5 apresenta um exemplo de como um usuário pode importar para o seu *SIG* um catálogo já existente na ferramenta *NDR-Tool*:

Figura 4 – Adicionando um catálogo existente a um SIG



Fonte: (ARAÚJO *et al.*, 2014)

A seguir a Tabela 2 apresenta as semelhanças e diferenças por tópico entre os trabalhos relacionados e o trabalho proposto.

Tabela 2 – Tabela comparativa entre os trabalhos relacionados.

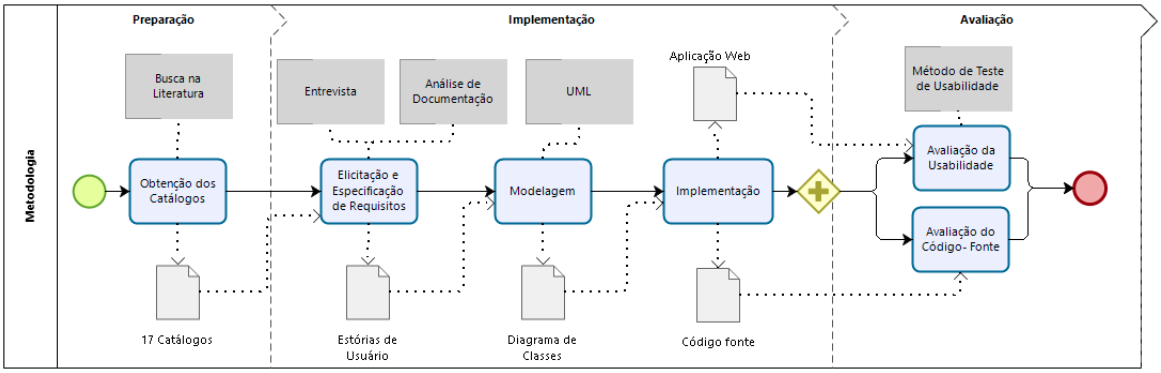
	NFR-Assistant	NDR-Tool	SIGFinder
Disponibilidade da ferramenta	Não está disponível em nenhuma plataforma	Disponível para a plataforma *desktop*	Desenvolvido para a plataforma *web*.
Suporte a adição de novos catálogos	Não possui catálogos	Não permite ao usuário adicionar catálogos	Projetado para permitir ao usuário adicionar catálogos que foram criados por ele
Disponibilidade de catálogos	Não disponibiliza catálogos na ferramenta	Disponibiliza os catálogos por tipo de RNF	Disponibiliza os catálogos por tipo de *RNF* e área de aplicação de *software*

Fonte: elaborada pelo autor

4 PROCEDIMENTOS METODOLÓGICOS

Nesta seção são apresentados os passos realizados e métodos utilizados para se atingir o objetivo deste trabalho. Cada sub-tópico possui uma descrição sobre o mesmo. A Figura 3 apresenta uma visão geral desses procedimentos.

Figura 5 – Procedimentos metodológicos



Fonte: elaborada pelo autor

4.1 Obtenção dos Catálogos

A primeira atividade que foi necessária para a criação do repositório foi a obtenção dos catálogos disponíveis. O objetivo do projeto é mostrar ao engenheiro de requisitos o máximo de catálogos disponíveis em qualquer domínio de aplicação ao qual ele precise de forma simples e rápida que traga o maior auxílio possível durante a atividade de priorização e escolha dos RNFs. Tendo isso em vista, foi realizada uma busca em artigos, livros, sites, entre outras fontes para que o maior número de catálogos pudesse ser obtido. A busca foi feita em duas bases de dados bastante utilizadas e difundidas entre os pesquisadores, são elas: Google Scholar [1] e Scopus[2]. Apenas os catálogos em formato de *SIG* foram armazenados, já que estes apresentam um nível mais específico de informações.

4.2 Elicitação e Especificação de Requisitos

Elicitação é o coração do desenvolvimento dos requisitos. É o processo onde você identifica as necessidades e restrições dos vários *stakeholders* do seu sistema de software

[1] https://scholar.google.com.br/
[2] https://www.scopus.com/home.uri

(WIEGERS K. ; BEATTY, 2013). Em outras palavras, elicitar é a atividade de receber o máximo de informações possíveis sobre os desejos do(s) cliente(s) a respeito do software.

Depois da obtenção dos catálogos, foi necessário que os requisitos fossem extraídos para que a modelagem do repositório pudesse ser criada a partir das informações obtidas. Existem várias técnicas de elicitação e especificação de requisitos, entre elas: entrevistas, workshops, grupos focais, e outras. As técnicas utilizadas para este projeto foi a Análise da Documentação e Entrevista.

A Análise da Documentação demanda a examinação de qualquer documento existente para potenciais requisitos de software (WIEGERS K. ; BEATTY, 2013). No âmbito deste projeto a documentação existente são os catálogos que foram obtidos anteriormente. Com uso destas técnicas de elicitação, os principais requisitos necessários para esse projeto foram identificados e documentados em forma de estórias de usuário [3] para que a modelagem do sistema fosse concebida.

A entrevista foi realizada em diversos encontros com a orientadora deste trabalho, cuja a área de pesquisa é voltada em catálogos de RNFs.

4.3 Modelagem

A modelagem do repositório foi baseada em diagrama UML(*Unified Modeling Language*) [4]. Nesta etapa do processo, os requisitos que foram elicitados, juntamente com as estórias de usuário, foram a base para que o diagrama de classe que é utilizado no repositório fosse criado.

4.4 Implementação

O próximo passo na construção do repositório foi a própria implementação. O repositório foi construído para a plataforma web. Por essa razão, alguns *frameworks* foram utilizados na concepção do *software*.

Em suma, o repositório foi desenvolvido em cima das tecnologias e frameworks a seguir: Spring Frameork [5], GoJS [6], Materialize [7], HTML, CSS e JavaScript. O banco de dados

[3] https://docs.google.com/spreadsheets/d/1$_M$C43Nx1IqSVQG71KCPdyGGLZ7657sE2TzYtrT3250k/edit?usp=sharing
[4] http://www.uml.org/
[5] https://spring.io/projects/spring-framework
[6] https://gojs.net/latest/index.html
[7] https://materializecss.com/

utilizado foi o H2 [8].

A Seção 4 abordará melhor como o repositório foi desenvolvido fazendo uso desses *frameworks*.

4.5 Avaliação do Repositório

A validação em um sistema de software é de grande importância pois ela obtêm dados dos usuários do seu sistema, a experiência deles, a fim de verificar se o que foi desenvolvido está de acordo com a necessidade que eles possuem.

A validação do repositório se deu por meio de um teste de usabilidade e de uma avaliação interna do próprio código fonte do *back-end* gerado da aplicação. Segundo (RUBIN JEFF.; CHISNELL, 2008), um teste de usabilidade utiliza-se de um conjunto de técnicas para obter dados empíricos enquanto usuários executam atividades reais no software. No teste que foi executado foram convidados alunos do curso de engenharia de software e profissionais de TI da Universidade Federal do Ceará - Campus Quixadá, que já concluíram a cadeira de requisitos de software com média superior ou igual a 7,0. A validação do repositório será tanto quantitativa quanto qualitativa. Para conseguir os dados quantitativos da avaliação, o seguinte conjunto de atividades são disponibilizados para os usuários:

- Buscar um catálogo por domínio;
- Buscar todos os catálogos de uma só vez;
- Visualizar um catálogo;
- Criar um catálogo

O teste foi executado por um avaliador, de forma que algumas medidas foram coletadas, são elas: tempo de conclusão por atividade, quantidade de erros para cada atividade e quantidade de vezes que o usuário recorreu à ajuda. Cada tarefa será melhor descrita na Seção de Avaliação do repositório.

O objetivo da avaliação foi ter um *feedback* dos usuários no que se refere ao quanto o repositório auxilia na forma de consultar os RNFs e na facilidade do seu uso. Com todas as informações que foram reunidas do teste usabilidade, melhorias poderão ser discutidas e realizadas no repositório.

[8] http://www.h2database.com/html/main.html

5 SIGFINDER - O REPOSITÓRIO ONLINE DE SIGS

Nesta seção será apresentada o repositório que foi desenvolvida neste projeto bem como suas principais funcionalidades.

5.1 Funcionalidades

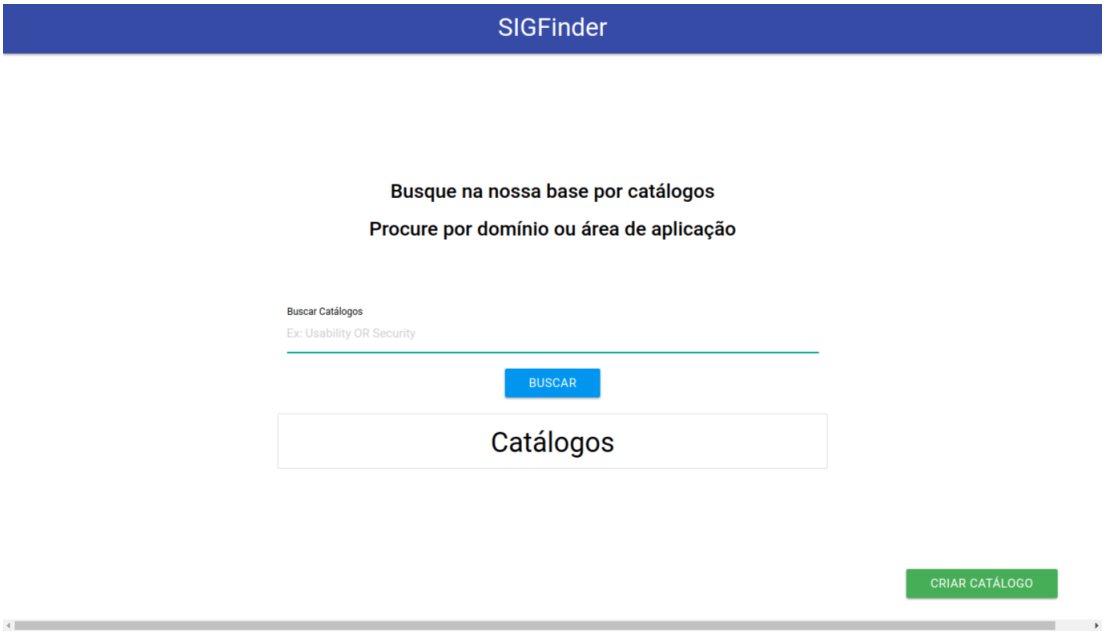
O repositório SIGFinder foi desenvolvido para a plataforma *web*. Algumas das estórias de usuário criadas na seção de levantamento de requisitos foram implementadas nesta primeira versão do repositório. As principais funcionalidades disponíveis nesta versão foram consideradas prioritárias para a solução que foi proposta por este trabalho, são elas:

- Buscar por todos os catálogos
- Buscar por catálogos com um domínio de aplicação específico
- Visualizar um catálogo disponível na repositório.
- Criar um SIG (*Softgoal Interdependecy Graph*).

O objetivo dessa solução é centralizar os catálogos disponíveis para facilitar o acesso a eles pelos desenvolvedores. Dessa forma, a interface do sistema foi pensada e projetada para que os usuários que desejam buscar, visualizar e criar catálogos alcançassem com êxito esses alvos de maneira simples e rápida. Nos próximos parágrafos serão exibidas *screenshots* da aplicação para cada funcionalidade citada anteriormente.

Na tela principal o usuário já pode ter acesso a maioria das funções disponíveis. A Figura 6 apresenta a página inicial do repositório:

Figura 6 – Tela Inicial - SIGFinder



Fonte: elaborada pelo autor

Os catálogos na ferramenta podiam ser encontrados conforme a *string* de busca que o usuário informasse na aplicação. Os SIGs estão organizados internamente na ferramenta com base em duas informações, qual o tipo de RNF (ex: *Usability*, *Security*, entre outros) e a plataforma a qual eles pertencem (ex: *Web*, *Mobile*, *Desktop*, entre outras).

Conforme o usuário informasse o tipo de RNF ou a plataforma do catálogo que ele desejava encontrar, essa *string* era enviada ao *back-end* da aplicação que buscava em todos os catálogos que estavam armazenados no banco de dados pelo tipo de RNF ou plataforma desejada do usuário. Caso a aplicação encontrasse algum catálogo, ela retornava a lista dos SIGs encontrados, apresentava-os na tela e informava ao usuário por meio de uma mensagem na tela a quantidade de catálogos encontrados. Se a busca não retornasse resultados, a ferramenta apresentava ao usuário uma mensagem informando que a busca não gerou resultados.

Para buscar os catálogos disponíveis no repositório, o usuário só precisava pressionar o botão “BUSCAR” no centro da tela sem passar nenhuma *string* de busca. Na data que os *screenshots* do repositório foram capturados, haviam sido adicionados ao banco de dados apenas 4 catálogos. A Figura 7, apresenta a tela da aplicação ao usuário realizar a busca:

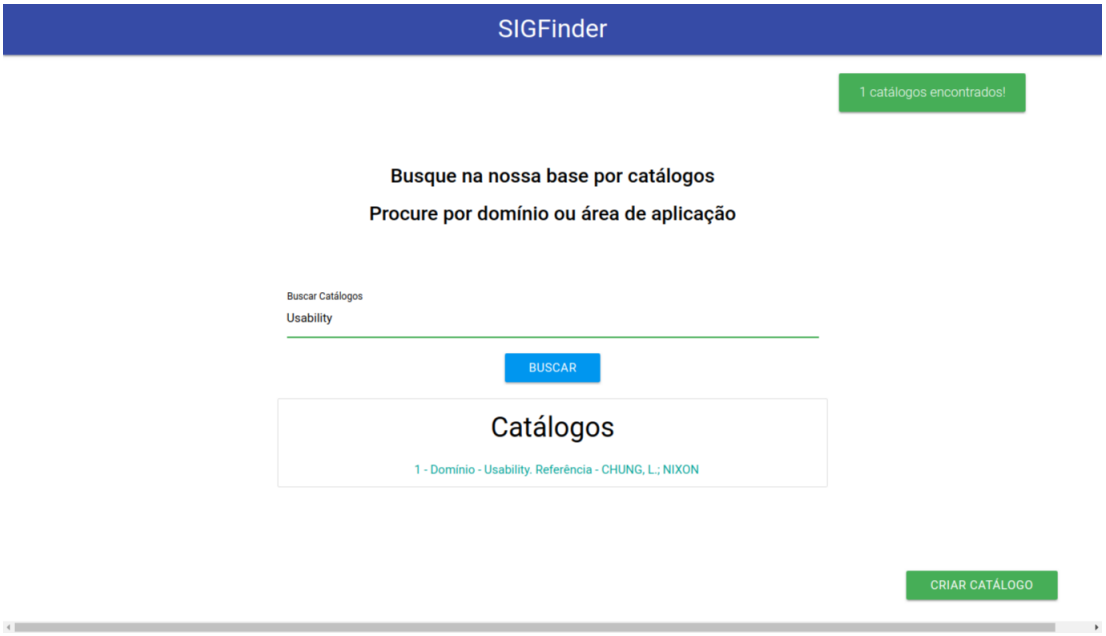
Figura 7 – Buscar todos os catálogos - SIGFinder

Fonte: elaborada pelo autor

Além da lista de catálogos disponíveis, o repositório também dá um *feedback*, no canto superior direito da tela, de quantos catálogos foram encontrados, esse retorno ao usuário aparece tanto para as buscas por todos os catálogos quanto quando o usuário deseja procurar por domínios específicos. A lista retornada pela aplicação retorna ao usuário, para cada catálogo, o domínio ao qual ele pertence e a referência do mesmo. Essa informação é de grande importância tendo em vista que haverão inúmeros catálogos que irão pertencer ao mesmo domínio ou área de aplicação dentro do repositório. Por meio da referência o usuário poderá discernir e escolher qual SIG ele deseja visualizar.

Para buscar por um catálogo de um RNF em específico, o usuário só necessita digitar o nome do RNF. A Figura 8 apresenta um exemplo de busca por tipo de RNF. Neste exemplo, o usuário digitou "*Usability*" e todos os catálogos existentes de Usabilidade são dispostos na tela para o usuário (no exemplo acima, apenas 1 catálogo era de "*Usability*").

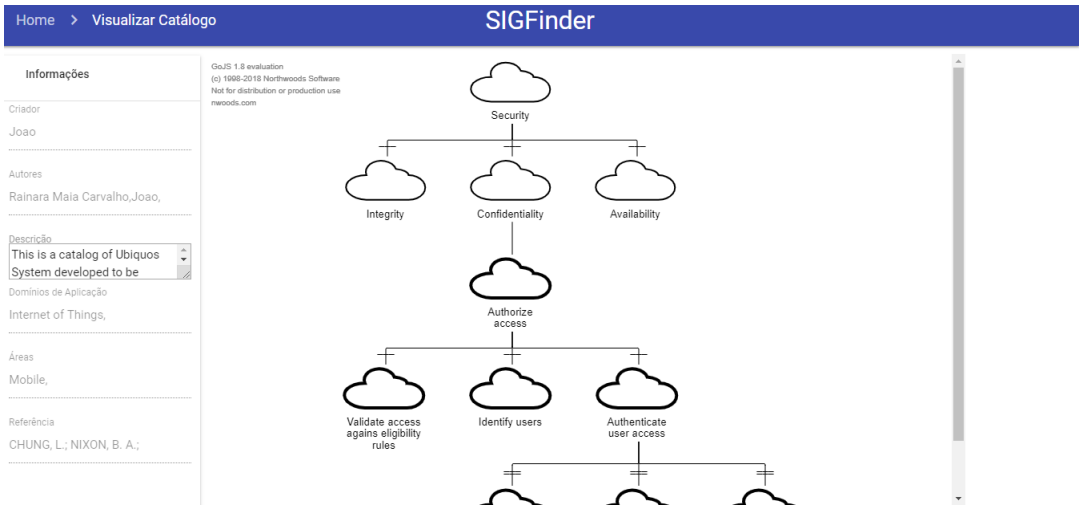
Figura 8 – Buscar catálogos por RNF - SIGFinder



Fonte: elaborada pelo autor

Para visualizar um catálogo, o usuário só necessita clicar em um dos itens da lista de catálogos mostrados a partir de uma busca realizada. A Figura 9 apresenta como visualizar um catálogo. Neste exemplo, o usuário selecionou um catálogo da lista e a seguinte tela foi exibida:

Figura 9 – Visualizar catálogo - SIGFinder



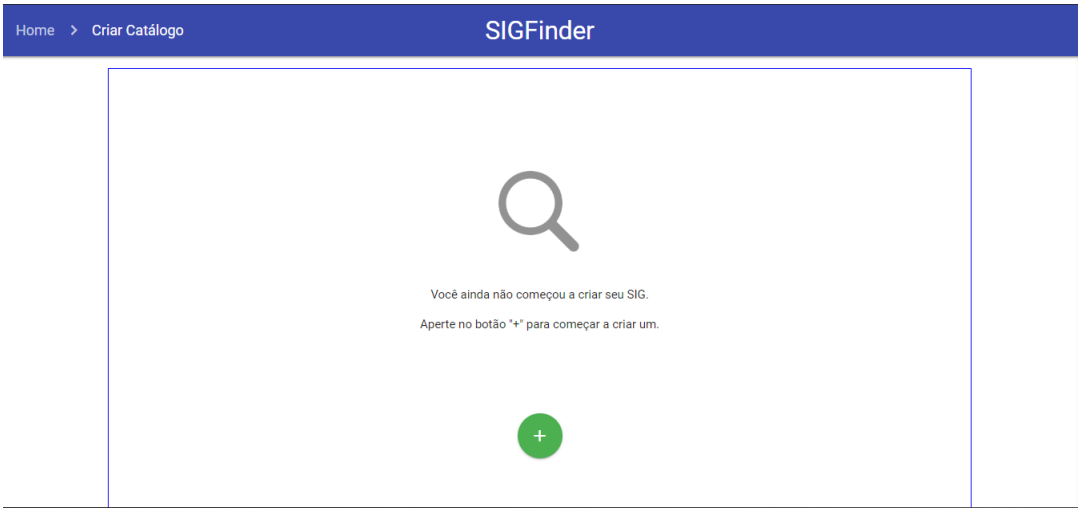
Fonte: elaborada pelo autor

Na tela de visualizar catálogo, não apenas o *SIG* em si é mostrado, mas também informações relevantes sobre ele, tais como: criador do catálogo (quem cadastrou o catálogo

na repositório), autores, descrição, domínios de plicação, áreas de aplicação e referência. Essas informações podem ser úteis no que diz respeito aos usuários entenderem um pouco do contexto ao qual o *SIG* foi criado. Note que o SIGFinder faz uso graficamente de vários conceitos propostos pelo *NFR Framework* (CHUNG *et al.*, 2000). Esses conceitos serão melhor abordados na próxima subseção deste trabalho.

A última funcionalidade que estava disponível na primeira versão da repositório é a de criar o seu próprio *SIG*. Para ter acesso a isto, o usuário só precisa clicar no botão "Criar Catálogo"que fica na página inicial da aplicação. Após essa ação, a seguinte tela será mostrada a ele como pode ser visto na Figura 10:

Figura 10 – Criar catálogo - SIGFinder



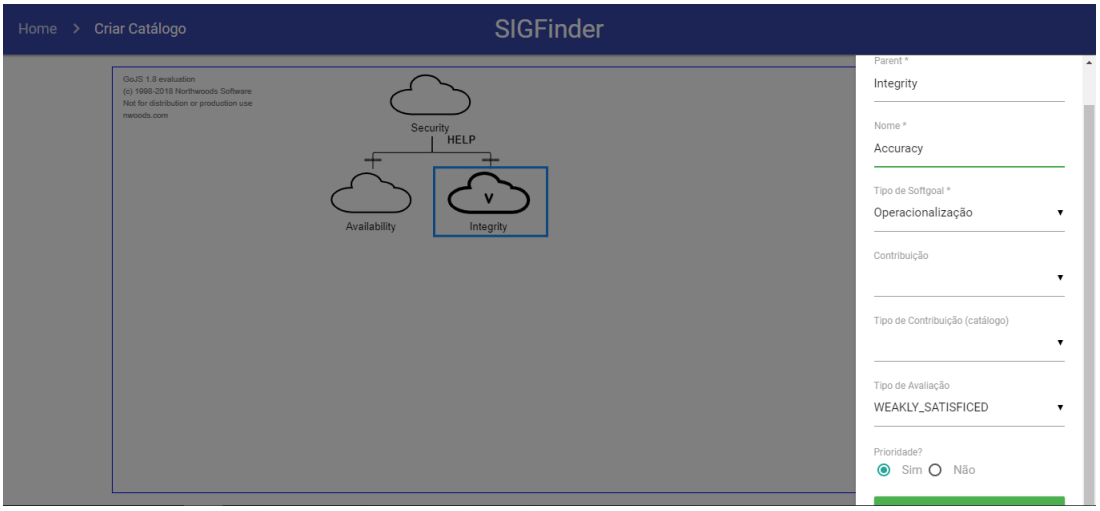
Fonte: elaborada pelo autor

Um *SIG* é constituído de um conjunto de *softgoals* que são interligados entre si (CHUNG *et al.*, 2000), assim sendo, o usuário para começar a criar seu *SIG* deverá clicar no botão verde que está no centro da tela na parte inferior. Ao clicá-lo, uma janela de navegação irá ser mostrada a ele na qual ele poderá colocar as informações cabíveis ao *softgoal* desejado, as informações possíveis são: nome, tipo de contribuição, tipo de contribuição de catálogo, tipo de procedimento de avaliação e se o *softgoal* é prioridade ou não no *SIG*. A Figura 11 apresenta isso em forma gráfica:

Figura 11 – Criar catálogo, primeiro *softgoal* - SIGFinder

Fonte: elaborada pelo autor

Figura 12 – Criar catálogo, adicionar *softgoal* - SIGFinder

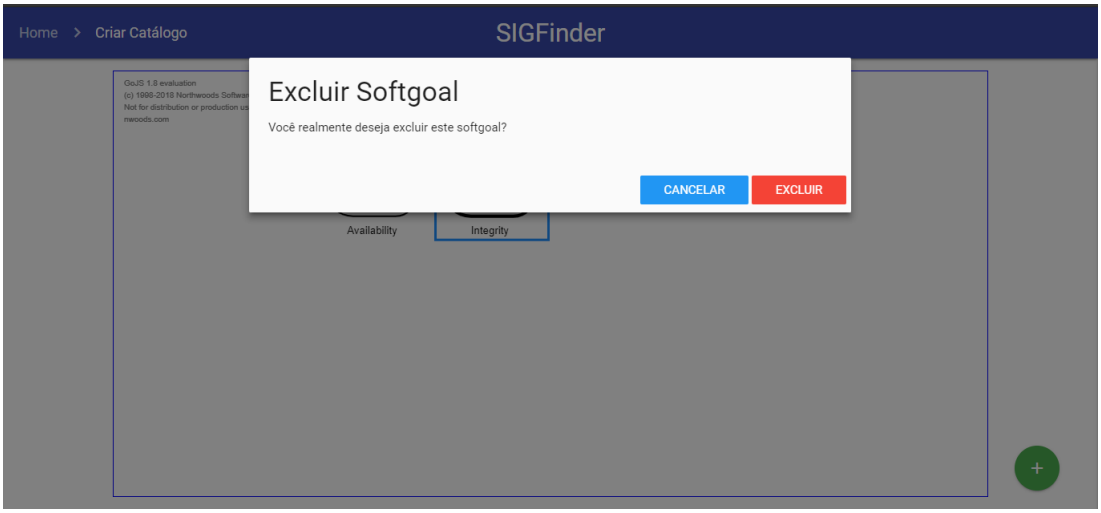


Fonte: elaborada pelo autor

Conforme pode ser visualizado na Figura 12, após criar o primeiro *softgoal* do catálogo, o usuário pode adicionar quantos *softgoals* desejar. Para tal, basta ele clicar em algum *softgoal* já criado e a mesma janela de navegação irá aparecer no lado direito da tela. Note que ao adicionar um *softgoal* a outro, o campo *parent* é mostrado ao usuário indicando qual é o *parent* vinculado ao *softgoal* que está sendo criado no momento. Além dele poder novamente configurar o *softgoal* que está sendo criado com tipos de contribuição, tipo de procedimento de avaliação e se o *softgoal* é prioritário ou não.

Para se excluir um softgoal o usuário apenas precisa deixar o mouse sobre o botão de adicionar um novo *softgoal* no canto inferior direito da tela e esperar um botão aparecer em cima do botão de adicionar, então ao clicar neste novo botão aparecia um *modal* no qual a aplicação pergunta ao usuário se ele realmente deseja excluir o *softgoal* selecionado, o seguinte cenário pode ser visualizado na Figura 13:

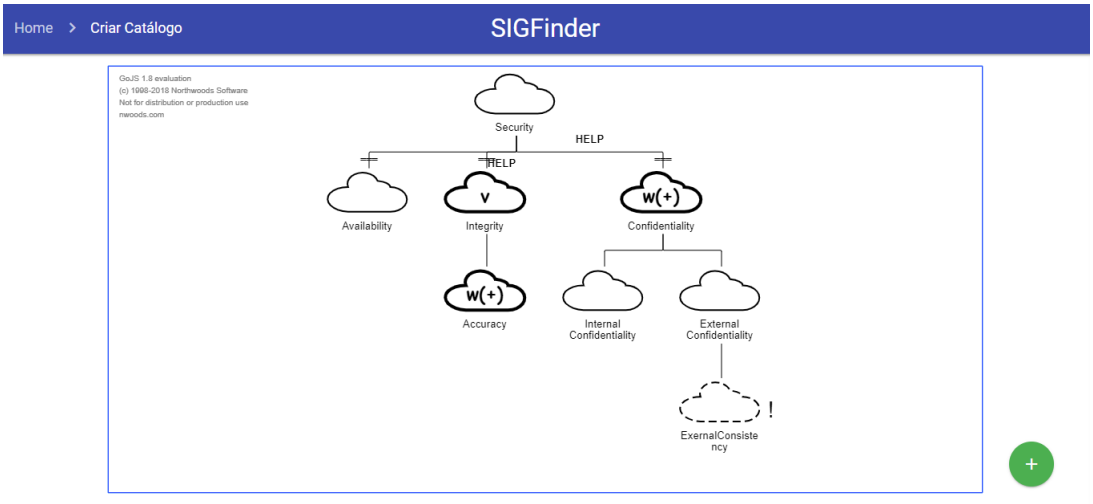
Figura 13 – Excluir *softgoal* - SIGFinder



Fonte: elaborada pelo autor

Por fim, um possível catálogo a ser criado pode ser visualizado na Figura 14. O catálogo em questão foi criado a partir de um exemplo contido em (CHUNG *et al.*, 2000).

Figura 14 – Exemplo de catálogo criado na aplicação - SIGFinder



Fonte: adaptado pelo autor de (CHUNG *et al.*, 2000)

Nas próximas seções maiores detalhes sobre as tecnologias e *frameworks* utilizados serão apresentados.

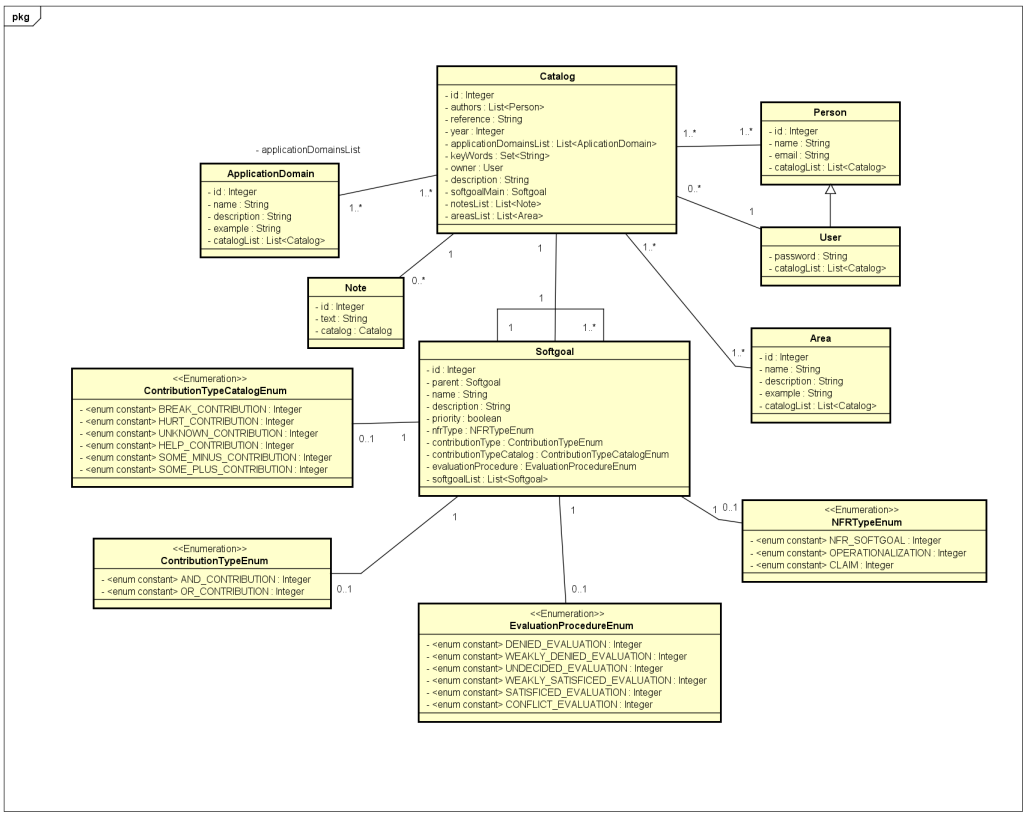
5.2 Modelagem do repositório

Durante o processo de modelagem, houveram vários impasses no que se refere a qual seria a melhor maneira de modelar o repositório. Na parte mais relacionada aos fundamentos teóricos, ou seja, o domínio de aplicação, algumas enumerações foram criadas para modelar os conceitos de tipos de *softgoal*, tipo de avaliação entre *softgoals*, contribuição dentro de um catálogo, entre outras. A classe *core* da aplicação é a "Catalog", ela é a responsável por representar um *SIG*. Dentro dela as informações principais de um catálogo de RNFs estão contidas, como: autores, descrição, referências, notas, entre outras e principalmente, o *softgoal* "principal" do catálogo. Esse softgoal principal pode contêr uma lista de outros *softgoals* que cada um deles por sua vez pode conter uma lista de outros *softgoals* e assim sucessivamente até chegar na "raiz" do catálogo gerado.

Por ser a fase da modelagem de um *software* não estática, ou seja, passível de mudança conforme haja necessidade, foram criadas até o presente momento, 2 (duas) versões do diagrama de classe criado. Essas versões podem ser encontradas no repositório da ferramenta no GitHub [1]. O diagrama de classe do repositório SIGFinder é apresentado na Figura 15.

[1] https://github.com/yuricavalcantedev/SIGFinder-backend/

Figura 15 – Diagrama de classe do repositório - SIGFinder



Fonte: criado pelo autor

5.3 *Back-end* da aplicação

A aplicação foi desenvolvida conforme descrito na Seção Metodologia deste trabalho. Durante a fase de desenvolvimento (que perdurou em paralelo a outras atividades) um diagrama de classe foi criado e atualizado conforme fosse notado que mudanças precisariam ser feitas nas classes criadas.

Uma boa prática de desenvolvimento de software é criar para cada projeto, um repositório *online* onde várias pessoas possam utilizar o código fonte e outros artefatos lá contidos, além da possibilidade de fazerem alterações nestes arquivos. Como SCV (Sistema de Controle de Versão), foi utilizada a repositório GitHub [2]. O repositório da parte *back-end* do SIGFinder pode ser encontrado no link contido no rodapé desta página [3].

[2] https://github.com
[3] https://github.com/yuricavalcantedev/SIGFinder-backend

5.4 *Front-end* da aplicação

Para o *front-end* da aplicação foram desenvolvidas páginas web em HTML, CSS e JavaScript. Para essa primeira versão, apenas 3 páginas foram desenvolvidas: a página inicial, a de visualizar um catálogo e a de criar um catálogo. Para auxiliar no desenvolvimento, foram utilizadas bibliotecas de *software* de terceiros que foram essenciais para a construção da repositório, são elas: Materialize [4] e GoJS [5].

O Materialize é um *framework* voltado para construção de páginas web que faz uso do *Material Design* da Google. Como descrito na seção de metodologia deste trabalho, o *Material Design* é um sistema adaptável de componentes, diretrizes e repositórios que fornece as melhores práticas voltadas ao design da interface de usuário [6].

Um dos objetivos ao se construir a aplicação era o foco na usabilidade e facilidade dos usuários ao se utilizá-la. Por isso usar um *framework* que já implementasse os melhores padrões de design que auxiliam na usabilidade do usuário, foi muito importante e poupou tempo de desenvolvimento. Além disso, a maioria dos componentes que são fornecidos pelo Materialize são responsivos, o que também ajuda na experiência do usuário com a aplicação.

O outro *framework* utilizado foi o GoJS. Ele é uma biblioteca *JavaScript* que fornece um conjunto de funcionalidades que auxiliam os desenvolvedores a criar diagramas e outras estruturas que possam mostrar dados de forma gráfica. Mais precisamente ele foi utilizado para construir os *SIGs* da aplicação.

A biblioteca disponibiliza uma *API* bem documentada e também vários códigos e diagramas de exemplos que auxiliaram na implementação dos *SIGs*.

5.4.1 Boas práticas de Desenvolvimento

Algumas boas práticas foram identificadas e utilizadas na aplicação desenvolvida que estão relacionadas a entrega contínua e rapidez na entrega do produto, influenciando assim a qualidade do *software*. Algumas dessas práticas utilizadas são: o uso de repositório de Integração Contínua (IC), Jenkins [7]; um analisador estático de código SonarQube [8] e um outra ferramenta responsável por fazer testes sistêmicos na aplicação (mais precisamente testes de carga) em

[4] https://materializecss.com/
[5] https://gojs.net/latest/index.html
[6] https://material.io/design/introduction/
[7] https://jenkins.io/
[8] https://www.sonarqube.org/

ambiente de homologação, ou seja, enquanto a aplicação já esteja disponível de forma *online*. Para tal teste a ferramenta JMeter [9] foi utilizada. Todas estas práticas estão associadas ao conceito de *DevOps*.

O termo *DevOps* veio de uma mistura entre desenvolvimento (no que se refere a programadores, testadores e pessoas que trabalham na qualidade do *software* e do pessoal envolvido) e de operações (representando as pessoas que colocam o '*software* no ar'). *DevOps* descreve práticas utilizadas para agilizar o processo de entrega de *software* e melhoria do mesmo (MICHAEL, 2012).

Essas atividades relacionadas a qualidade do *software* asseguram um nível de padronização no qual os desenvolvedores precisam atingir a cada versão nova gerada do sistema. Atualmente existe apenas um desenvolvedor trabalhando na aplicação, porém conforme o repositório evolua e mais pessoas participem do time de desenvolvimento, essas configurações geradas irão assegurar a qualidade do código e do *software* desenvolvido.

[9] https://jmeter.apache.org/

6 AVALIAÇÃO DO SIGFINDER

Depois da ferramenta ter sido implementada, foi realizado um teste de usabilidade para verificar e mensurar quão fácil é utilizá-la e qual o nível de satisfação dos usuários ao utilizá-la. Além disso, várias técnicas e abordagens foram dirigidas à ferramenta buscando a qualidade interna do código gerado. A seção abaixo apresenta detalhadamente como o teste de usabilidade foi preparado e executado.

6.1 Teste de Usabilidade

Este teste foi desenvolvido seguindo as diretrizes fornecidas por (BARBOSA SIMONE D.J; SILVA, 2010). O teste foi dividido em três etapas: Preparação, Execução e Resultados. Todas essas etapas serão descritas nas sub seções abaixo.

6.1.1 Preparação

O teste de usabilidade tem como objetivo avaliar a usabilidade de um sistema interativo a partir da experiência de seus usuários alvos (BARBOSA SIMONE D.J; SILVA, 2010). Cada teste de usabilidade pode diferir de outro, mesmo que seja para o mesmo sistema, isso ocorre quando os objetivos da avaliação desse sistema são diferentes (BARBOSA SIMONE D.J; SILVA, 2010).

No teste realizado para o SIGFinder, o objetivo principal era descobrir quão fácil era utilizar o sistema, ou seja, o alvo era medir a facilidade que os usuários tinham para executar determinadas tarefas. As três principais respostas que o teste em questão deveria fornecer eram:

- Qual a quantidade de erros cometida pelos usuários a cada tarefa?
- Quantas vezes os usuários recorrem à ajuda disponível a cada tarefa?
- Quanto tempo os usuários levam para finalizar cada tarefa?

Para o presente teste foram elaborados 4 documentos para recolherem informações dos usuários bem como auxiliarem durante o teste. Os documentos gerados foram: Termo de Consentimento (Apêndice 1), cada usuário assinou este termo permitindo assim a captura dos dados que fossem gerados durante o teste; entrevista pré-teste (Apêndice 2), cada usuário que participou do teste respondeu essa pequena entrevista que tinha como objetivo capturar informações demográficas e também recolher informações sobre o domínio de cada um deles no que se refere a área de requisitos não-funcionais; o terceiro documento criado foi o Cenário

de Tarefas (Apêndice 3), que tinha como finalidade informar a cada usuário sobre como o teste iria ocorrer, quais técnicas que seriam utilizadas, como por exemplo, a técnica *Think aloud*, que é uma técnica utilizada para que o usuário sempre expresse de forma verbal as reações e sentimentos que ele tem enquanto está interagindo com o sistema e também guiá-lo durante o teste, informando cada atividade que deveria ser realizada; e por último o Questionário pós-teste (Apêndice 4), onde várias perguntas (qualitativas e quantitativas) foram feitas para saber como havia sido a interação de cada usuário com o sistema (baseado em: Questionário de avaliação Usabilidade -SUS [1]).

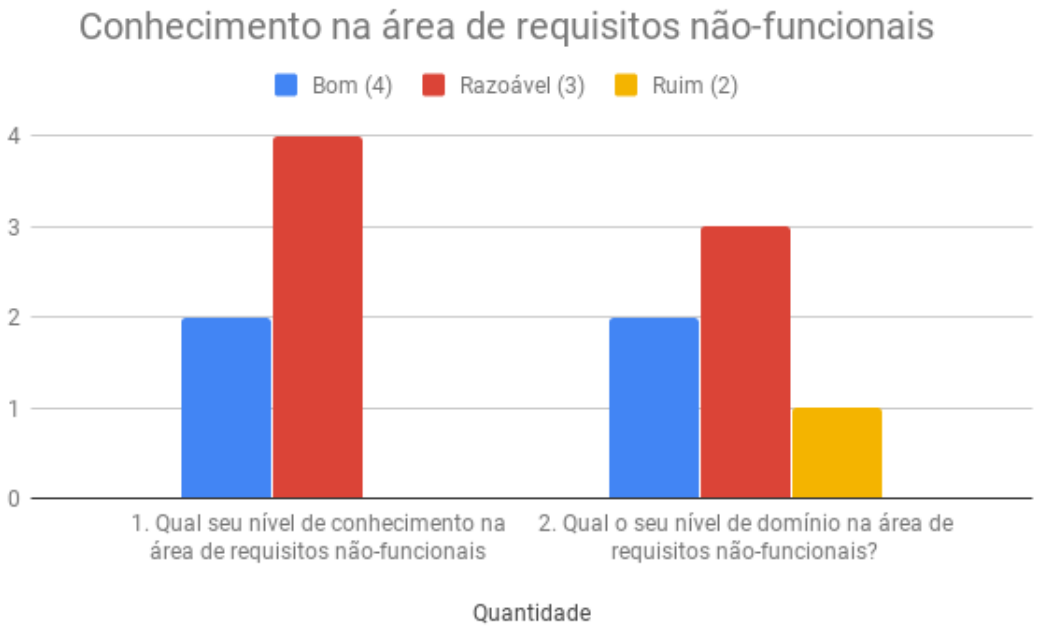
Foram convidados para fazerem parte desse teste 6 pessoas que estudavam ou trabalhavam na Universidade Federal do Ceará - Campus Quixadá. Ao todo eram 4 estudantes, sendo um deles também profissional na área de designer, 1 técnico te tecnologia da informação e outro analista de tecnologia da informação. Foram 3 mulheres e 3 homens que participaram do teste. A média de idade entre eles era de 24,3 anos. Também foi perguntado a eles a experiência que tinham em desenvolvimento de software, a média encontrada para essa pergunta foi 3,6 anos.

Além dos dados demográficos, também foi perguntado a cada um deles qual o conhecimento e domínio na área de requisitos não-funcionais e se já haviam trabalhado com SIGs e criado um SIG anteriormente.

Para a questão relacionada ao conhecimento sobre RNFs, 33,36% dos participantes responderam que possuíam um bom conhecimento sobre RNFs enquanto 66,64% responderam que tinham um conhecimento razoável sobre o assunto. Já em relação a dominar a área de requisitos não-funcionais, 33,32 % responderam que possuíam um bom domínio sobre a área, 49,98% assertaram que os seus conhecimentos eram razoáveis e apenas 16,7% responderam que o nível de conhecimento era ruim. Os dados podem ser visualizados na Figura 16:

[1] https://www.maxwell.vrac.puc-rio.br/29090/29090$_9$.*PDF*

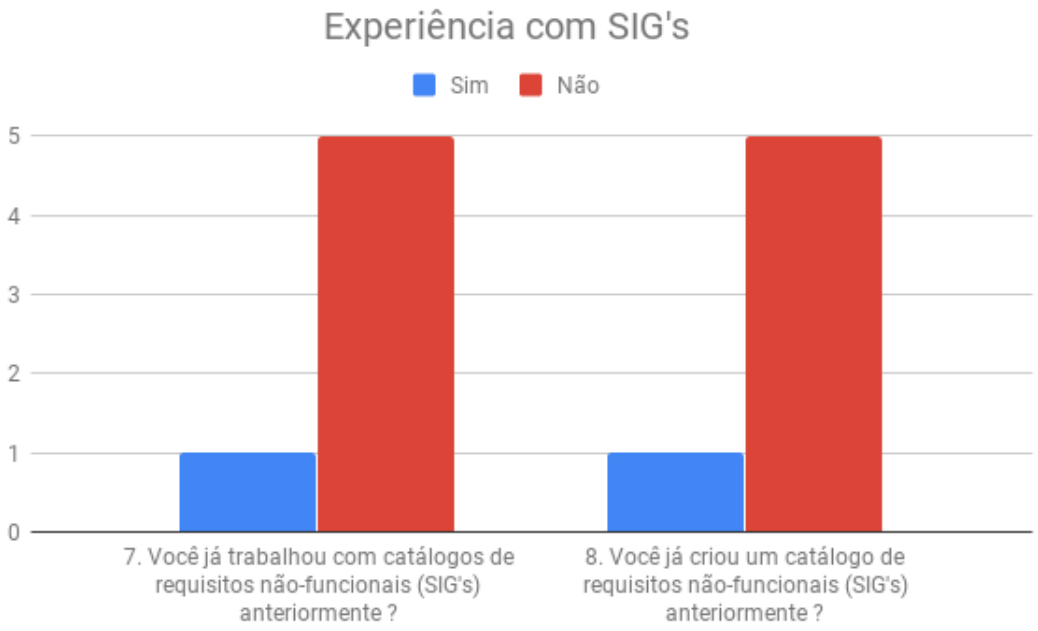
Figura 16 – Perguntas Relacionadas a Requisitos Não-Funcionais



Fonte: elaborada pelo autor

No que se refere a ter experiência com SIGs, foram mostradas 2 perguntas para os usuários. Na primeira, 83,3% responderam que nunca haviam trabalhado com SIGs enquanto apenas 16,7% afirmaram terem trabalhado com SIGs. Na segunda pergunta, os mesmos resultados foram obtidos, ou seja, 83,3% responderam que nunca haviam criado um SIG, enquanto 16,7% dos usuários afirmaram já terem criado um SIG. Os dados podem ser visualizados na Figura 17:

Figura 17 – Experiência com Requisitos Não-Funcionais



Fonte: elaborada pelo autor

Para que todos os usuários pudessem ter o nível de conhecimento sobre RNFs e SIGs nivelados, foi criada uma apresentação utilizando o Google Apresentações [2] e então foi exibida para os usuários. Foram 3 exibições em momentos diferentes. A primeira foi com 4 usuários em conjunto e mais outras duas, uma para cada usuário que não pôde comparecer à exibição em grupo. À apresentação e todos os documentos citados nesta seção podem ser encontrados no link que se encontra no rodapé desta página. [3]

Para que não houvesse interferência na execução de cada usuário, o teste ocorreu em uma sala reservada onde apenas o avaliador (autor deste projeto) e o usuário estavam presentes.

6.1.2 Execução

Os testes ocorreram durante os dias 28 e 29/11 deste ano. O avaliador do teste preparava o local do teste e convidava cada usuário para entrar na sala e então realizar a execução do teste.

Antes do teste ser executado, cada participante assinava o termo de consentimento e respondia ao questionário pré-teste. Após isso, o avaliador entregava o documento de Cenário

[2] https://docs.google.com/presentation/u/0/
[3] https://drive.google.com/drive/folders/1T1mRyvSynrEqqjQ–yEHRHWLOWOfuppS

de Tarefas (Apêndice C) para os participantes e explicava alguns pontos que seriam necessários para a execução do teste. Entre estes pontos, era informado ao participante que um áudio seria gravado durante o teste para que informações pudessem ser analisadas e extraídas em um momento posterior.Também era explicado de maneira rápida sobre o que consistia a técnica *Think aloud* e então era solicitado ao participante que fizesse uso da mesma. Além disso era informado ao participante que ele poderia fazer uso de 2 tipos de ajuda no decorrer do teste: ele poderia utilizar ou à apresentação que foi utilizada para "mostrar"mais sobre o domínio de requisitos não-funcionais e SIGs, ou então perguntar ao avaliador do teste. Por último, era pedido ao participante que a cada tarefa ele dissesse em voz alta uma frase semelhante a esta: 'Estou iniciando a tarefa X' e no final da atividade: 'estou finalizando a atividade x'. Isso foi requisitado para que não fosse obtida apenas a informação de quanto tempo durou o teste, mas também de quanto tempo cada atividade levou para ser realizada, para que uma das três perguntas levantadas na seção anterior pudesse ser respondida, a saber, 'Quanto tempo cada atividade leva para ser executada?'.

Durante a execução do teste, o avaliador fazia as marcações de tempo cada vez que o participante iniciava ou finalizava uma atividade proposta e contabilizava a quantidade de erros cometidos em cada tarefa bem como a quantidade de vezes que o usuário buscou ajuda em cada tarefa. Durante a realização do primeiro teste, foi-se notado que a tarefa 3 não poderia ser alcançada, por esse motivo, do participante 2 em diante ela foi cancelada.

No primeiro teste já perto da conclusão, nos últimos 7 minutos, alguns alunos da faculdade entraram na sala e ficaram lá conversando, o que acabou atrapalhando um pouco o participante. No final o avaliador do teste perguntou ao participante se a interferência dos alunos havia sido muito prejudicial, o usuário disse que sim, mas não muito. No decorrer dos outros testes não houve interferência de terceiros.

Ao longo dos testes o avaliador auxiliou no entendimento de algumas das tarefas, porém não foram contabilizadas como ajuda na execução da atividade.

Após a conclusão das 6 tarefas restantes do teste, cada participante respondeu ao questionário pós-teste (Apêndice D). Os resultados obtidos do questionário pós-teste podem ser encontrados na sub-seção abaixo.

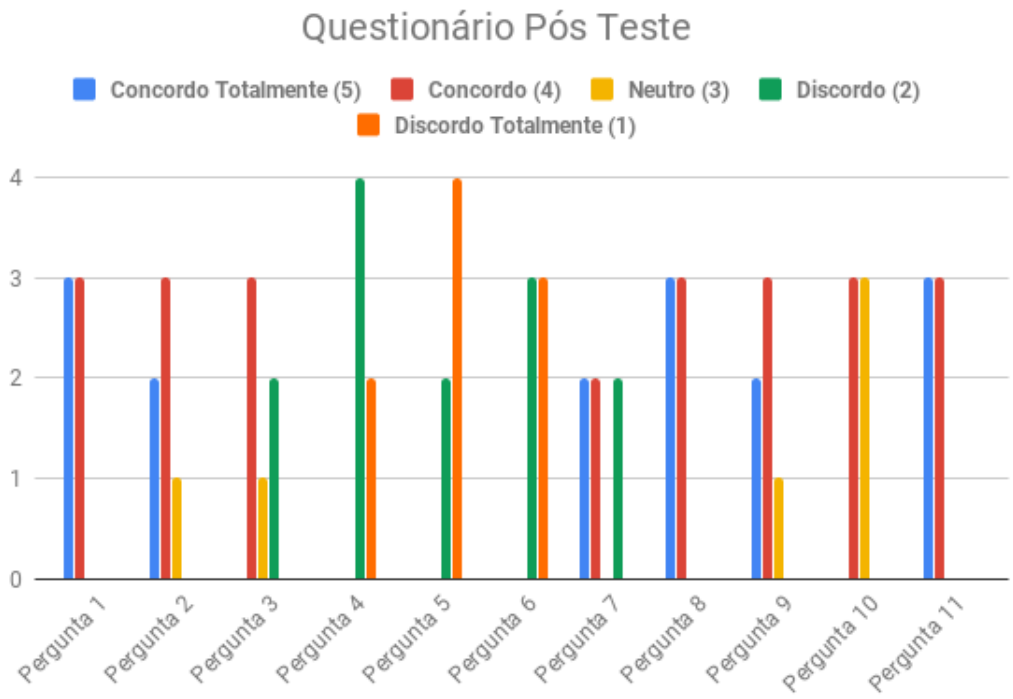
6.1.3 Resultados

O questionário pós teste era formado por 11 (onze) perguntas que poderiam ser respondidas marcando uma opção das disponíveis (perguntas quantitativas) e por 3 perguntas abertas nas quais cada participante do teste poderia descrever pontos positivos, negativos e também informar melhorias ou sugestões sobre o sistema.

6.1.3.1 Perguntas Objetivas

Um gráfico foi gerado com as respostas das 11 perguntas. O mesmo pode ser visualizado na Figura 18 (as perguntas na legenda se referem ao Questionário pós-teste contido no Apêndice D deste trabalho):

Figura 18 – Questionário Pós Teste



Fonte: elaborada pelo autor

Como pode ser visto no gráfico, o nível de satisfação com o sistema foi bom. Nas perguntas relacionadas a facilidade de uso do sistema e sobre conforto ao utilizar o sistema, 83,32% dos participantes responderam que concordavam (totalmente ou apenas concordavam) que o sistema era fácil de se usar e 100% dos usuários afirmaram que se sentiram confortáveis

(em relação a facilidade de uso e esforço) ao utilizar a ferramenta.

Também foi perguntado se o usuário gostaria de utilizar o sistema posteriormente caso houvesse necessidade e se ele indicaria para algum colega de trabalho a ferramenta SIGFinder, as respostas respectivamente para essas perguntas foram: 100% dos usuários concordaram que eles utilizariam novamente o SIGFinder caso houvesse necessidade e essa mesma taxa de porcentagem também afirmaram que poderiam indicar a ferramenta para algum colega de trabalho.

No que se refere a perguntas relacionadas a facilidade de uso para completar 3 tarefas específicas do Cenário de Tarefas (buscar um catálogo, visualizar um catálogo e criar um catálogo), as seguintes respostas foram inferidas: 66,64% dos participantes discordaram que foi difícil fazer a busca de um catálogo e os outros 33,36% discordaram totalmente dessa afirmação; a mesma divisão das respostas foi encontrada quando se perguntado se visualizar um catálogo havia sido difícil, 33,36% responderam que discordavam de tal assertiva e 66,64% replicaram que discordavam totalmente de tal afirmação; e por fim, 100% dos participantes discordaram (incluindo totalmente) que houvesse sido difícil criar um catálogo.

Quando se perguntado sobre a interface do sistema e sobre a disposição das informações na tela, os participantes responderam da seguinte maneira: 16,66% permaneceram neutros quanto a afirmação da interface do sistema ser agradável enquanto 83,34% concordaram (incluindo totalmente) que a interface era agradável. Além disso, 50% dos participantes se posicionaram de forma neutra quanto as informações estarem bem organizadas na tela enquanto os outros 50% concordaram que as informações estavam bem organizadas.

Por fim, ainda outros dados podem ser extraídos desse gráfico, como: 50% dos participantes afirmaram que as funções do sistema estavam explícitas e 66,64% responderam que precisa ter um prévio conhecimento da área de RNFs para utilizar o sistema.

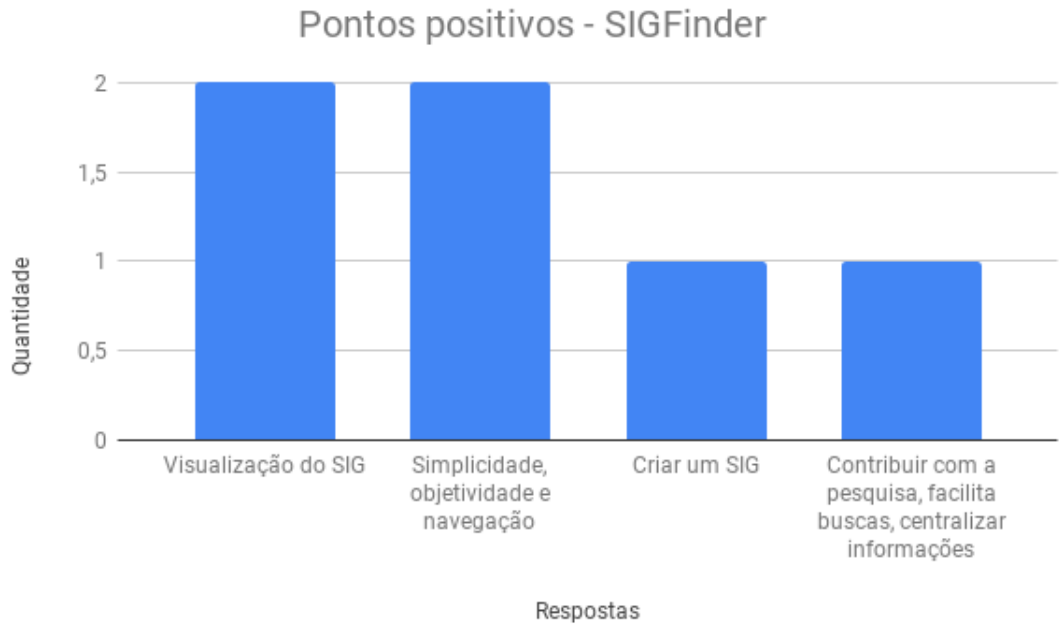
Como os dados demonstram, em sua maioria os participantes acharam fácil utilizar o sistema, bem como realizar as 3 tarefas principais do SIGFinder. Muitos afirmaram que utilizariam de novo o sistema e até recomendariam para colegas de trabalho que necessitassem da aplicação. Esses dados demonstram que, por mais que hajam pontos a melhorar (como será visto nas subseções abaixo), a ferramenta tem um bom nível de usabilidade e satisfação dos usuários que não apenas realizaram o teste proposto, mas podem em um futuro próximo serem reais usuários fazendo uso do SIGFinder, e os catálogos dentro dele contidos, dentro de empresas e corporações nas quais venham a trabalham buscando soluções para problemas reais de RNFs.

6.1.3.2 Perguntas Subjetivas

Embora houvessem perguntas subjetivas no questionário, foi possível sumarizar as respostas de cada pergunta, para os pontos positivos, pontos negativos e sugestões e melhorias.

A Figura 19 apresenta as respostas dos usuários sobre um ponto positivo na ferramenta.

Figura 19 – Pontos Positivos - SIGFinder



Fonte: elaborada pelo autor

Como pode ser inferido do gráfico, 33,32% dos participantes apontaram como positiva a visualização do SIG, nesse contexto, foi bastante satisfatório ouvir e ver, durante a execução do teste, usuários se admirando com a interface do sistema e como o catálogo estava disposto na ferramenta. A mesma porcentagem de usuários respondeu como ponto positivo da ferramenta, a simplicidade e objetividade da navegação.

Por fim, 16,66% dos usuários apontaram como ponto positivo para a ferramenta tanto a criação de um SIG quanto a contribuição da ferramenta para a pesquisa, pois facilitava a busca de catálogos e centralizava as informações.

Os resultados obtidos dessa pergunta também ajudam a validar a conclusão obtida na seção anterior no que se refere a usabilidade do SIGFinder. Também foi muito satisfatório um

dos usuários perceber que o SIGFinder tem uma contribuição para a área de pesquisa de RNFs.

Já em relação aos pontos negativos da ferramenta os usuários relataram, em geral, as respostas que podem ser visualizadas na Figura 20:

Figura 20 – Pontos a melhorar - SIGFinder

Fonte: elaborada pelo autor

Por ser uma pergunta subjetiva, podia-se relatar mais de um problema no sistema, essa é a razão de terem sido coletadas 7 respostas através desta pergunta. O gráfico mostra que 33,32% dos usuários relatou como ponto negativo do sistema a falta de um espaço onde pudessem ser editados os softgoals que estavam sendo criados pelos usuários. De fato, essa função não havia sido implementada a data do teste de usabilidade. A mesma porcentagem de usuários (33,32%) respondeu que o botão excluir estava não estava tão intuitivo na interface.

Por fim, 3 outros pontos negativos tiveram a mesma porcentagem de usuários (16,66%) relatando-os, foram eles: detalhes da interface (de maneira mais geral), textos de ajuda nos botões (*tootips*) e o campo de buscar um catálogo estava 'escondido e não muito intuitivo'. Realmente, não haviam textos de ajuda nos botões que continham apenas uma imagem dentro dele. E a respeito do campo de busca, a imagem de *background* da página inicial pode ter atrapalhado na visibilidade do campo de buscar catálogo.

Os resultados obtidos através dessa pergunta mostra que embora o nível de satisfação

com o sistema tenha sido bom, ainda existem pontos a melhorar e a aperfeiçoar para melhorar ainda mais a experiência de usuário e a usabilidade dele na ferramenta. Dados assim são muito importantes pois dão uma direção de onde existem falhas e onde esforços devem ser concentrados para melhorar a ferramenta.

Por fim, na última pergunta do questionário, os usuários podiam dissertar sobre sugestões e melhorias para serem implementadas na ferramenta SIGFinder. As respostas em forma de gráfico podem ser visualizadas na Figura 21:

Figura 21 – Sugestões e Melhorias - SIGFinder

Fonte: elaborada pelo autor

Como último item do questionário, foi solicitado ao participante que propusesse sugestões e melhorias para a ferramenta. As respostas que os usuários forneceram foram todas relevantes para a melhoria da ferramenta, algumas até se repetiram com outras respostas da parte de pontos negativos (botão editar, visibilidade do botão excluir e textos de ajuda nos botões).

Várias funcionalidades que não haviam sido pensadas anteriormente foram sugeri-

das pelos participantes, tais como: duplicar um softgoal (6,66%), retirar os botões flutuantes (6,66%),criar um novo botão apenas para a procurar por todos os catálogos de uma única vez (13,33%) e no momento que for buscar por catálogos, considerar letras minúsculas e maiúsculas, além de considerar na pesquisa apenas uma parte da palavra e não apenas ela toda (13,33%).

Houveram respostas que já haviam sido ponderadas, porém não houve tempo suficiente para serem implementadas, são elas: botão de ajuda no próprio sistema (13,33%), textos de ajuda nos botões (13,33%) e melhorar a visibilidade do botão de pesquisa na página inicial.

Através dessa última pergunta foi possível validar funções que já haviam sido cogitadas pelo autor, mas que não houve tempo suficiente durante a execução desse trabalho para implementá-las, além de verificar a necessidade de melhoria em alguns pontos da interface que irão auxiliar ainda mais cada usuário em sua interação com o sistema.

6.1.3.3 Conclusões Gerais dos Resultados

No começo desta seção, foram pontuadas 3 perguntas que iriam auxiliar na medição da usabilidade da ferramenta proposta por esse trabalho. As perguntas eram:

- Qual a quantidade de erros cometida pelos usuários a cada tarefa?
- Quantas vezes os usuários recorrem à ajuda disponível a cada tarefa?
- Quanto tempo os usuários leva para finalizar cada tarefa?

Durante a execução de cada teste, o avaliador coletou informações que pudessem ser utilizadas para responder a cada pergunta dessas.

Gráficos foram gerados com os dados coletados para responder a cada pergunta. A Figura 22, apresenta quantos erros foram cometidos pelos usuários durante a execução de cada tarefa:

Figura 22 – Erros cometidos por atividade - SIGFinder

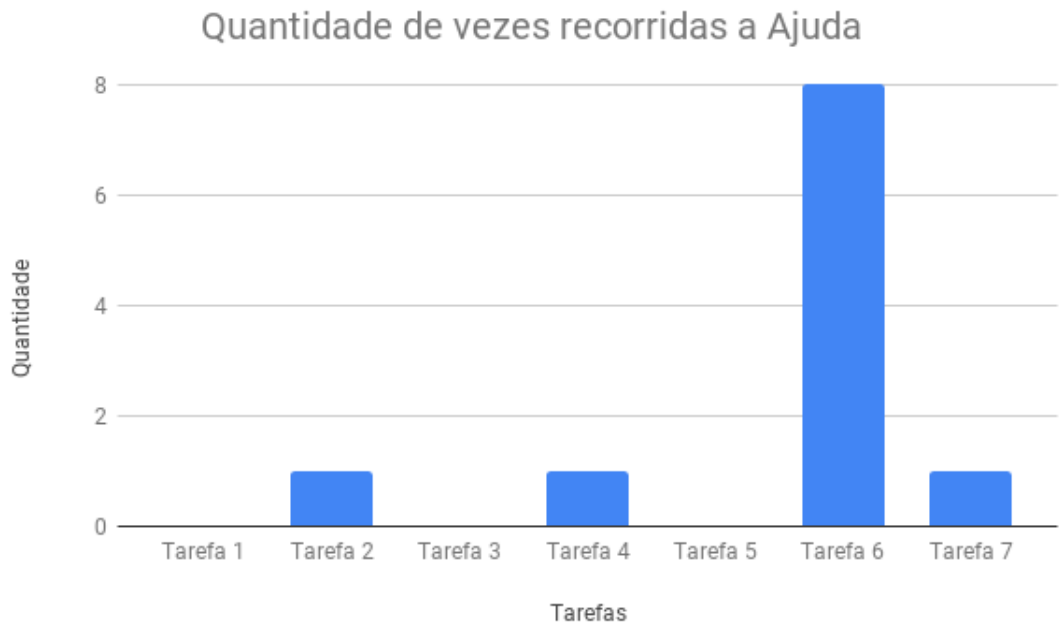
Fonte: elaborada pelo autor

Os dados apresentados na Figura 21 revelam alguns pontos importantes sobre algumas atividades. Como pode-se inferir, a Tarefa 1 foi a que os usuários mais cometeram erros durante o teste, foram 9 erros cometidos. Estes erros foram efetuados por 50% dos usuários (3 participantes), cada um deles errou 3 vezes até conseguir completar essa tarefa. Os motivos dessa taxa de erro podem ser vinculados as melhorias sugeridas pelos participantes, que são justamente sobre o campo de pesquisa ser mais visível e sobre o campo de busca não permitir letra minúsculas ou parte da palavra.

Outro dado importante de ressaltar é a quantidade de erros cometidos na Tarefa 2,4 e 5. Essas atividades tiveram o menor índice de erros durante o teste, foram respectivamente, 1 erro, 0 erro e 1 erro. É valido destacar esses dados pois essas tarefas são muito importantes no contexto da ferramenta, que tem como objetivo justamente disponibilizar catálogos para os desenvolvedores possam buscar (Tarefa 2 e 4) e visualizá-los (Tarefa 5).

Outro dado importante para responder as perguntas propostas pelo teste é a quantidade de vezes que os usuários recorreram a ajuda durante o teste. A Figura 23 apresenta os dados obtidos:

Figura 23 – Quantidade de Vezes que utilizaram ajuda - SIGFinder

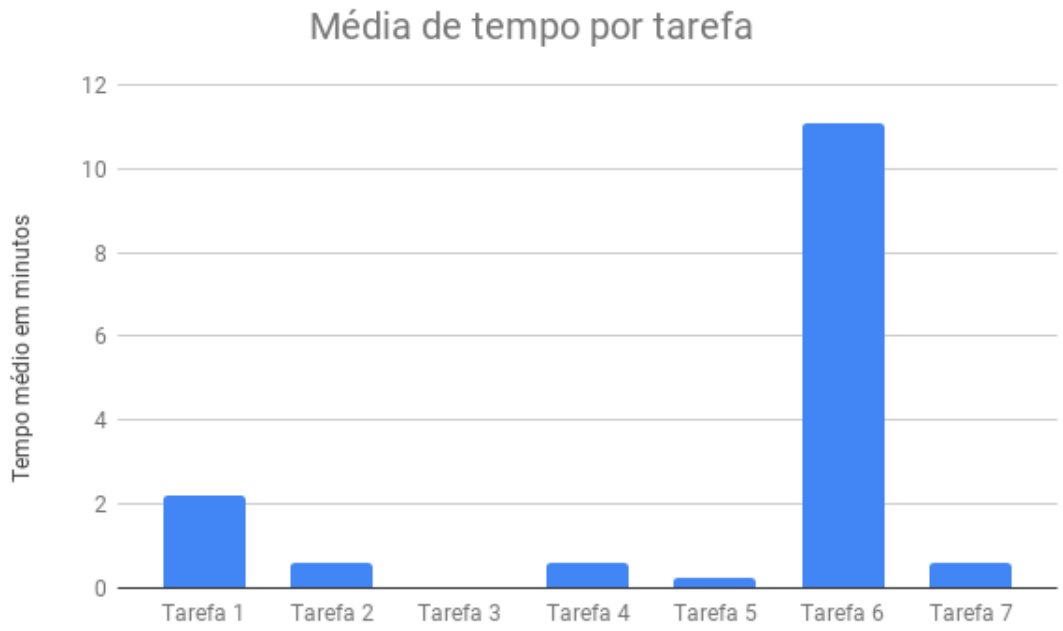


Fonte: elaborada pelo autor

Por meio dos dados apresentados na Figura 22 pode-se notar que pouco foi utilizada a ajuda disponível no teste, nas tarefas 1 e 5 nenhum usuário solicitou ajuda e nas tarefa 4 e 7 apenas um participante utilizou-se da ajuda.

Além disso, é importante ressaltar que na Tarefa 6 a ajuda foi utilizada 8 vezes. Esse número mais elevado já era esperado por conta do tamanho da Tarefa 6 e também por 83,6% dos usuários terem relatado que nunca haviam criado um SIG anteriormente. Incluso a isto também tem o fato de que foi pedido na Tarefa 6 para os usuários criarem um SIG que utilizasse todos os conceitos que haviam sido apresentados a eles na apresentação feita que tinha como objetivo nivelar o nível de conhecimento dos usuários sobre RNFs e catálogos. Mesmo com tudo isto, o tempo necessário para concluir esta Tarefa demonstrou que eles tiveram facilidade em construir um SIG. Os tempos médios de cada Tarefa é apresentado na Figura 24:

Figura 24 – Quantidade de Vezes que utilizaram ajuda - SIGFinder



Fonte: elaborada pelo autor

Das 6 tarefas disponíveis no teste, 4 delas tiveram tempo médio de execução menor do que 40 segundos, são elas: tarefa 2 (Procurar por um catálogo utilizando a string de busca "Internet of Things", 36.s segundos), a tarefa 4 (buscar por todos os catálogos contidos na ferramenta, 36 segundos, 14 segundos).a tarefa 5 (visualizar qualquer catálogo disponível na ferramenta) e a tarefa 7 (excluir um softgoal do catálogo que foi criado na tarefa 6, 36.2 segundos).

As tarefas que mais demoraram para serem completadas foram a tarefa 1 (pesquisar por um catálogo utilizando a *string* de busca "Usability", 1 minuto e 13 segundos) e a tarefa 6 (criar um catálogo com um conjunto específico de softgoals e contribuições, 11 minutos e 5 segundos).

Mesmo a tarefa 1 sendo a segunda tarefa mais demorada a ter sido realizada, o tempo dela ainda pode ser considerado bom. Um dos fatores que possa ter influenciado na relativa demora ao realizá-la, é o campo de busca não estar tão visível quanto deveria estar. A tarefa que mais levou tempo médio para ser executada foi a tarefa 6. Ela consistia em criar um catálogo com no mínimo 5 softgoals e várias outras restrições que podem ser lidas e encontradas no Apêndice C.

Dos dados acima, pode-se corroborar que, embora hajam algumas correções e melhorias a serem realizadas no SIGFinder, os usuários ficaram satisfeitos com a usabilidade da ferramenta.

6.2 Avaliação da Qualidade Interna do Código

Durante a fase de desenvolvimento da ferramenta, não apenas o código-fonte foi gerado, mas também houve a preocupação em mensurar a qualidade do código fonte, mais precisamente na parte de *back-end*. Como citado na seção anterior deste trabalho, um analisador estático foi integrado na ferramenta de integração contínua, que por meio deste, o código da aplicação foi avaliado com base nos seguintes critérios estabelecidos:

- *Bugs*, quantos *bugs* foram encontrados na aplicação
- Linhas duplicadas, a quantidade de linhas duplicadas no código
- Linhas comentadas, quantas linhas haviam sido deixadas comentadas dentro do código
- Vulnerabilidades, no que se refere, a visibilidade dos métodos dentro da ferramenta.
- *Code Smells*, é um termo utilizado na literatura para apontar que existem partes no código que não estão sendo implementadas da melhor maneira.

Após a análise do código, os dados a seguir foram encontrados (vale ressaltar que os dados aqui apresentados se referem as classes *core* da aplicação, essa informação é necessária pois o resultado de uma classe não foi contabilizado nos dados a seguir, pois a mesma continha métodos para adicionar catálogos de forma *hardcoded* na aplicação). Apenas 2 *bugs* foram identificados, que eram de classe C, ou seja, *bugs* de nível *major*. Os dois eram de possíveis *NullPointerException* que não possui nenhum tratamento de erro.

No que se refere as duplicações de linhas de código, a ferramenta de análise encontrou 247 linhas duplicadas. Embora o número seja alto, a explicação para este alto número é que os mesmos erros foram cometidos em classes ou *enums* semelhantes. Por exemplo, a aplicação possui 4 *enums* implementados. Então cada erro será contabilizado 4 vezes pelo analisador.

A quantidade encontrada de linhas comentadas foi de 40. Várias delas foram esquecidas durante o desenvolvimento da aplicação. Embora o número possa não ser grande, o alvo é deixar a aplicação com nenhuma linha comentada.

Para a métrica de vulnerabilidade, apenas 8 foram encontradas. 2 vulnerabilidades estavam presentes em 4 *enums*. A taxa de vulnerabilidade que a ferramenta indicou foi B. Isso significa que eram todas vulnerabilidades do tipo *minor*. Em uma próxima versão estas

vulnerabilidades deverão ser sanadas.

Por último, também foi analisada a quantidade de *code smells* da ferramenta. O analisador estático apontou 35 *code smells*. Destes, a maioria era do tipo *minor* (25), seguido de *major* (9) e info (1). É um número relativamente baixo. Mesmo assim, na próxima versão do SIGFinder, todos deverão ser corrigidos.

7 CONCLUSÕES

7.1 Contribuições

Percebendo-se a importância que catálogos de RNFs tem e a descentralização dos mesmos, que estão disponíveis nas mais variadas fontes, o presente trabalho propôs a criação de um repositório que disponibilizasse aos desenvolvedores e engenheiros de requisitos, a maior quantidade de catálogos possível em uma mesma aplicação. Diferentemente das duas ferramentas abordadas nos trabalhos relacionados deste projeto, a aplicação que é proposta (SIGFinder) foi desenvolvida para a plataforma *web*, facilitando assim o acesso a mesma.

Por meio dos testes de usabilidade realizados, a aplicação desenvolvida pôde ser avaliada com usuários-alvo reais. Por intermédio dos resultados obtidos, foi possível mensurar e avaliar a facilidade e usabilidade da aplicação criada. Os usuários demonstraram um bom nível de satisfação com a ferramenta, tanto no que tange a facilidade de uso, quanto na usabilidade do sistema, por meio da satisfação dos usuários e da eficiência deles ao completarem as tarefas.

Assim sendo, tendo em vista que a maioria dos usuários afirmou ter conhecimento razoável da área de requisitos não-funcionais e que nunca tiveram tido experiência com *SIGs*, por meio dos resultados obtidos, pode-se inferir que quando a ferramenta for disponibilizada na plataforma *web*, os usuários que têm um maior domínio sobre *RNFs* e *SIGs* irão ter semelhante ou superior nível de satisfação ao fazer uso do SIGFinder.

7.2 Lições Aprendidas

Embora o reuso das bibliotecas descritas neste trabalho tenha facilitado o desenvolvimento da repositório, várias dificuldades foram encontradas e consequentemente lições foram aprendidas durante a construção do repositório.

No que se refere ao *back-end* da aplicação, as dificuldades encontradas foram voltadas mais a qualidade do código e a *DevOps*. Fazer com que, através de um *commit* feito no repositório do código, todo o processo de *build*, de análise estática de código e a execução e relatórios, tanto de testes unitários quanto testes sistêmicos, acontecesse, foi bastante trabalhoso. Porém, por haver uma demanda crescente nas empresas de TI por profissionais que tenham conhecimento sobre a área de *DevOps*, foi muito proveitoso construir esse *background* da aplicação.

Embora problemas na parte do desenvolvimento relacionada ao código tenha ocorrido, pesquisas na *Internet* era executadas para auxiliar a solucionar estas dificuldades. Por meio dessas buscas na *Internet*, várias dúvidas foram solucionadas e assim o conhecimento sobre a linguagem Java foi aprimorado.

Por fim no *front-end*, o obstáculo que se sobressaiu aos outros foi o de mostrar graficamente o *SIG*. O *framework* que foi utilizado para isso não possuía nenhum diagrama que representasse graficamente um catálogo de *RNFs*. Por isso levou muito tempo até conseguir exibir cada *softgoal* em forma de nuvem na aplicação e mostrar a ligação entre eles. Foi bastante proveitoso trabalhar com uma biblioteca de um terceiro e desafiador também pois praticamente apenas a documentação e os exemplos da própria biblioteca estavam disponíveis, mas no fim foi muito bom ter esse desafio. Possuir a documentação como praticamente única fonte de pesquisa, certamente melhorou a leitura de documentação de bibliotecas, e esse é um conhecimento valioso.

7.3 Trabalho Futuro

Como trabalhos futuros, existem ainda muitos catálogos disponíveis nas bases de dados e livros que não foram adicionados à ferramenta. Portanto, mais catálogos deverão ser disponibilizados para a consulta dos usuários. Com base nas sugestões de melhoria que os usuários dos testes de usabilidade forneceram, algumas funcionalidades serão ajustadas.

Existem ainda estórias de usuário que não são contempladas na ferramenta na presente data atual, com base nelas outras funcionalidades (por exemplo: cadastro e login dos usuários e a opção do usuário poder favoritar catálogos, facilitando assim o acesso aos *SIGs* que ele precisa), serão desenvolvidas, testadas e adicionadas ao SIGFinder. A parte *back-end* da aplicação ainda está sendo utilizada apenas localmente. Um dos próximos passos é hospedá-la em um servidor *web* e disponibilizá-la para o uso dos usuários.

REFERÊNCIAS

ARAÚJO, A. L. de; CYSNEIROS, L. M.; WERNECK, V. **NDR-Tool** : uma ferramenta de apoio ao reuso de conhecimento em requisitos não funcionais. In: WER. [S.l.], 2014.

BARBOSA SIMONE D.J; SILVA, B. S. **Interação humano-computador**. Campus, 2010. ISBN 9780735679665. Disponível em: <https://www.saraiva.com.br/interacao-humano-computador-3064227.html>. Acesso em: 25 dez. 2018.

BERANDER, P.; DAMM, L.-O.; ERIKSSON, J.; GORSCHEK, T.; HENNINGSSON, K.; JÖNSSON, P.; KÅGSTRÖM, S.; MILICIC, D.; MÅRTENSSON, F.; RÖNKKÖ, K. *et al.* In: **Software quality attributes and trade-offs**: Blekinge institute of technology. [S.l.: s.n.], 2005.

CARVALHO, R. M.; ANDRADE, R. M. C.; OLIVEIRA, K. M.; KOLSKI, C. In: **Catalog of invisibility requirements for ubiComp and IoT applications**: Requirements engineering conference (re). [S.l.: s.n.], 2018.

CHUNG, L.; NIXON, B. A.; YU, E.; MYLOPOULOS, J. **Non-functional requirements in software engineering**. [S.l.]: Springer Science & Business Media, LLC, 2000. ISBN 9781461374039.

CYSNEIROS, L. M. **Requisitos Não Funcionais** : da elicitação ao modelo conceitual. Dissertação (Tese de doutorado) — PUC/RJ, 2001.

HEERY, R.; ANDERSON, S. **Digital repositories review**, Joint Information Systems Committee, 2005.

LAPOUCHNIAN, A. **Goal-oriented requirements engineering** : an overview of the current research, University of Toronto, p. 32, 2005.

MAIA, M. E.; ROCHA, L. S.; ANDRADE, R. **Requirements and challenges for building service-oriented pervasive middleware**: Proceedings of the 2009 international conference on pervasive services. In: ACM. [S.l.], 2009. p. 93–102.

MAIRIZA, D.; ZOWGHI, D. In: SPRINGER. **Constructing a catalogue of conflicts among non-functional requirements**: International conference on evaluation of novel approaches to software engineering. [S.l.], 2010. p. 31–44.

MICHAEL, H. **DevOps for developers**. Apress, 2012. ISBN 9781430245704. Disponível em: <https://www.apress.com/br/book/9781430245698#aboutAuthors>. Acesso em: 12 abr. 2018.

RUBIN JEFF.; CHISNELL, D. **Handbook of usability testing** : how to plan, design, and conduct effective tests. [S.l.]: Wiley Publishing, Inc, 2008. Second Edition. ISBN 9780470185483.

SOMMERVILLE, I. **Engenharia de Software, 9. Ed.** Pearson Education, 2011. ISBN 9788579361081. Disponível em: <https://www.amazon.com/Engenharia-software-Portuguese-Ian-Sommerville-ebook/dp/B00KDOM4VG>. Acesso em: 13 abr. 2018.

SUPAKKUL, S.; CHUNG, L. **The RE-tools** : a multi-notational requirements modeling toolkit. In: Requirements Engineering Conference (RE), 2012 20th IEEE International. [S.l.], 2012. p. 333–334.

TRAN, Q. T.; CHUNG, L. **A case tool for the non-functional requirements framework**. Dissertação (Mestrado) — University of Texas at Dallas, 1998.

TRAN QUAN.; CHUNG, L. In: **NFRAssistant** : tool support for achieving quality. [S.l.: s.n.], 1999.

WIEGERS K. ; BEATTY, J. **Software Requirements,** 3rd ed. Pearson Education, 2013. ISBN 9780735679665. Disponível em: <https://www.safaribooksonline.com/library/view/software-requirements/9780735679658/>. Acesso em: 25 abr. 2018.

YRJÖNEN, A.; MERILINNA, J. Nfpindsml@ models. In: **Extending the NFR framework with measurable nonFunctional requirements.** [S.l.: s.n.], 2009.

APÊNDICE A – TERMO DE CONSENTIMENTO

TERMO DE CONSENTIMENTO - TESTE DE USABILIDADE

SIGFINDER

DADOS DE IDENTIFICAÇÃO

Título do projeto: SIGFinder

Instituição do aluno: Universidade Federal do Ceará - Campus Quixadá

Professor responsável: _____

Avaliador do teste: _____

Nome do voluntário: _____

Idade:_____ Profissão: _____

Estado Civil: _____

Data: _____/_____/_____

O presente documento tem como finalidade obter o consentimento do envolvido acima citado para fazer uso de suas informações que forem extraídas antes (entrevista pré-teste), durante (teste de usabilidade da ferramenta) e depois (questionário pós-teste) do teste de usabilidade da ferramenta SIGFinder.

Eu_____, declaro ter sido informado e concordo em participar do teste de usabilidade da ferramenta SIGFinder, da entrevista pré teste e do questionário pós-teste – bem como liberar a utilização, por parte do avaliador do teste, de meios eletrônicos (aúdio, vídeo e fotos) e escritos (anotações) com o intuito de registrar minhas informações para fins acadêmicos e de pesquisa – para o projeto de pesquisa acima descrito, a saber, SIGFinder.

_____

Assinatura do usuário

_____

Assinatura do avaliador

APÊNDICE B – QUESTIONÁRIO PRÉ-TESTE

Questionário pré-teste - SIGFinder

Nome do Usuário: _____

Avaliador: _____

Data do Teste: ____/____/____

Perguntas

Demográficas

1. Qual a sua idade? _____

2. Qual o seu sexo? _____

3. Qual sua profissão? _____

4. Quanto tempo de experiência você tem em profissão? _____

Área de Requisitos Não-Funcionais

*Observação: Você pode marcar apenas uma resposta por pergunta.

	Alto (5)	Bom (4)	Razoável (3)	Ruim (2)	Insuficiente (1)
5. Qual seu nível de conhecimento na área de requisitos não-funcionais	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
6. Qual o seu nível de domínio na área de requisitos não-funcionais?	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1

7. Você já trabalhou com catálogos de requisitos não-funcionais (SIG's) anteriormente ?

☐ Sim ☐ Não

8. Você já criou um catálogo de requisitos não-funcionais (SIG's) anteriormente ?
(Ignore essa pergunta caso a resposta anterior seja não)

☐ Sim ☐ Não

APÊNDICE C – CENÁRIO DAS TAREFAS

Cenários das Tarefas - Teste de Usabilidade SIGFinder

Nome do usuário: _____

Avaliador: _____

Data: _____/_____/_____

Informações

Este documento tem como finalidade descrever quais as tarefas que deverão ser realizadas no teste de usabilidade. O teste em questão tem como objetivo validar a interface que foi desenvolvida para a ferramenta SIGFinder, bem como conseguir dados quantitativos e qualitativos para que possam ser analisados e posteriormente possam haver melhorias na ferramenta.

Instruções Gerais

- Ao início e ao final de cada tarefa você deverá falar: “Irei iniciar a tarefa” e “Tarefa Finalizada”, respectivamente.
- Faça uso da técnica “Think Aloud”, ou seja, sempre “pense em voz alta” aquilo que você quiser expressar ao utilizar o sistema.
- Você poderá consultar a qualquer momento a “Apresentação do Teste de Usabilidade” para tirar possíveis dúvidas que surgirem do contexto da aplicação.

As seguintes tarefas deverão ser realizadas:

Tarefas

Tarefa 1:

Procure por um catálogo com domínio de aplicação “Usability”.

Tarefa 2:

Procure por um catálogo com domínio de aplicação “Internet of Things”.

Tarefa 3:

Procure por catálogos que possuam ou o domínio de aplicação “Usability” ou sejam da área de “Mobile”.

Tarefa 4:

Procure por todos os catálogos existentes na ferramenta.

Tarefa 5:

Visualize qualquer catálogo disponibilizado na ferramenta.

Tarefa 6:

Crie um novo catálogo. Este catálogo deverá ter o seguinte formato:

- Possuir 5 SOFTGOALS
- Possuir 3 níveis de SOFTGOALS
- Possuir 2 softgoals com TIPO DE SOFTGOAL (NFR_SOFTGOAL)
- Possuir 2 softgoals com TIPO DE SOFTGOAL(OPERATIONALIZATION)
- Possuir 1 softgoals com TIPO DE SOFTGOAL(CLAIM)
- Possuir ao menos 2 softgoals com TIPO DE CONTRIBUIÇÃO (AND ou OR).
- Possuir ao menos 2 softgoals com TIPO DE CONTRIBUIÇÃO DE CATÁLOGO (BREAK, HURT, UNKNOWN, HELP, SOME_MINUS, SOME_PLUS)
- Possuir ao menos 3 softgoals com TIPO DE AVALIAÇÃO (DENIED, WEAKLY_DENIED, UNDECIDED, WEAKLY_SATISFICED, SATISFICED, CONFLICT)
- Possuir ao menos 2 softgoals com PRIORIDADE

Tarefa 7:

Excluir um Softgoal do último nível do catálogo.

APÊNDICE D – QUESTIONÁRIO PÓS TESTE

Questionário Pós Teste - SIGFinder

Nome do Usuário: _____

Avaliador: _____

Data do Teste: ____/____/____

*Observação: Você pode marcar apenas uma resposta por pergunta.

	Concordo Totalmente (5)	Concordo (4)	Neutro (3)	Discordo (2)	Discordo Totalmente (1)
1. Gostaria de usar esse sistema caso houvesse necessidade	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
2. O sistema é fácil de usar	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
3. As funções deste sistema estavam explícitas	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
4. Fazer a busca de um catálogo foi muito difícil	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
5. Visualizar um catálogo foi muito difícil	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
6. Criar um catálogo foi muito difícil	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
7. É necessário ter um prévio conhecimento para se usar o sistema	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
8. Me senti confortável ao utilizar o sistema	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
9. A interface do sistema é agradável	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
10. As informações estavam bem organizadas nas telas do sistema	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1
11. Recomendaria o uso desse sistema para colegas de trabalho	☐ 5	☐ 4	☐ 3	☐ 2	☐ 1

Para você, qual é um ponto positivo do sistema?

_____

Para você, qual é um ponto negativo do sistema?

_____

Quais melhorias ou sugestões você pode dar para melhorar a usabilidade do sistema?