Revista Eletrônica de Sistemas de Informação
ISSN 1677-3071

v. 14, n. 3

set-dez 2015

doi:10.21529/RESI.2015.1403

Sumário

Nominata de avaliadores

Avaliadores ad hoc - 2015

Esta revista é (e sempre foi) eletrônica para ajudar a proteger o meio ambiente, mas, caso deseje imprimir esse artigo, saiba que ele foi editorado com uma fonte mais ecológica, a *Eco Sans*, que gasta menos tinta.

This journal is (and has always been) electronic in order to be more environmentally friendly. Now, it is desktop edited in a single column to be easier to read on the screen. However, if you wish to print this paper, be aware that it uses Eco Sans, a printing font that reduces the amount of required ink.

UMA TAXONOMIA UNIFICADA PARA REQUISITOS NÃO FUNCIONAIS

A UNIFIED TAXONOMY FOR NON FUNCTIONAL REQUISITES

(artigo submetido em agosto de 2013)

Fabiane Barreto Vavassori Benitti
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
fabiane.benitti@ufsc.br; fabiane.benitti@gmail.com

Jaqueline Sezra Rhoden
Pós-graduanda em Qualidade e Engenharia de Software na
Universidade do Vale do Itajaí (UNIVALI)
jaqueline.rhoden@gmail.com

ABSTRACT

The usefulness of a system is determined by its functionality and also by its non-functional characteristics such as security, reliability, usability, performance, among others. However, these "non-functional" characteristics typically receive little attention and are less understood that the functional ones and, in addition, the literature contains several classifications of non-functional requirements (RNF), making it more difficult the understanding of this type of requirement. This paper aims to propose a taxonomy for non-functional requirements (NFR) developed by unifying diverse taxonomies in the existing literature. The proposed taxonomy covers 90 RNF aspects, and can be used as a guideline during the elicitation of user requirements, a way of organizing the requirements, facilitating the verification and validation and even reuse of these requirements.

Key-words: non-funcional requirements; taxonomy.

RESUMO

A utilidade de um sistema é determinada pela sua funcionalidade e, também, por suas características não funcionais, tais como confiabilidade, segurança, usabilidade, desempenho, entre outras. No entanto, essas características não funcionais normalmente recebem pouca atenção e são menos compreendidas do que as funcionais e, além disso, a literatura apresenta diversas classificações de requisitos não funcionais (RNF), dificultando ainda mais o entendimento desse tipo de requisito. O presente artigo tem por objetivo propor uma taxonomia para requisitos não funcionais (RNF) elaborada por meio da unificação de diversas taxonomias existentes na literatura. A taxonomia proposta abrange 90 aspectos relacionados com RNFs, podendo ser utilizada como um guia durante a elicitação de requisitos; uma forma de organizar os requisitos, facilitando a verificação e validação e, até mesmo, a reutilização desses requisitos.

Palavras-chave: requisitos não funcionais, taxonomia.

1 INTRODUÇÃO

Nas últimas décadas, a crescente demanda por sistemas de softwares mais complexos e com melhor qualidade vem exigindo que o mercado de software atenda não apenas as funcionalidades requeridas, mas também a aspectos não funcionais tais como: segurança, confiabilidade, portabilidade, entre outros. Esses aspectos devem ser tratados como requisitos não funcionais (RNF) do software.

Os RNF estão raramente associados às características individuais do sistema. Pelo contrário, esses requisitos especificam ou restringem as propriedades emergentes do sistema. Portanto, eles geralmente são mais importantes do que os requisitos funcionais individuais (SOMMERVILLE, 2007). Por isso, esses requisitos necessitam ser capturados e analisados desde os primeiros momentos do desenvolvimento do software. Erros provocados por não se lidar convenientemente, ou simplesmente não se lidar com RNF, são apontados entre os mais caros e difíceis de corrigir (CYSNEIROS, 2001).

Por outro lado, os requisitos não funcionais têm recebido pouca atenção e são menos compreendidos do que outros fatores referentes ao desenvolvimento de software (CHUNG; LEITE, 2000). Além disso, quando citados nos processos de desenvolvimento, estes requisitos são abordados de maneira secundária e altamente informal do ponto de vista da elicitação (BASTOS, 2007). Outro ponto importante, já evidenciado em Chichinelli (2002), é que não foi encontrada na literatura uma lista exaustiva dos diversos aspectos não funcionais de um projeto de software. Ao invés disso, diferentes taxonomias de requisitos não funcionais foram propostas para auxiliar no processo de identificação dos RNF (SOMMERVILLE, 2007; ISO/IEC 25010, 2010; IEEE, 1998; RATIONAL SOFTWARE, 2002; CONALLEN, 2002; WITHALL, 2007 e LEFFINGWELL, 2007). Observa-se ao analisar as categorias de RNF propostas (e apresentadas na seção 2) que não há um consenso entre os autores, dificultando o entendimento deste tipo de requisito. Portanto, a unificação dessas classificações proporcionaria uma taxonomia mais robusta, que poderia auxiliar os engenheiros de sistemas durante a elicitação, rastreamento e verificação dos requisitos não funcionais.

Assim, este artigo tem como objetivo propor uma taxonomia para RNF baseada na unificação de diversos tipos de requisitos não funcionais propostos na literatura, caracterizando esta pesquisa quanto ao seu objetivo como uma pesquisa descritiva. O procedimento adotado para esta finalidade foi a pesquisa bibliográfica que, segundo Gil (2007, p. 44), “se propõe à análise das diversas posições acerca de um problema”. Para esta finalidade foram considerados livros específicos da área de requisitos de software ou de desenvolvimento, não considerando, neste momento, artigos, teses ou outras fontes. Adicionalmente, foi realizada uma pesquisa documental, envolvendo normas e processos de desenvolvimento propostos na literatura. Por fim, considera-se esta uma pesquisa de aplicada, pois objetiva gerar conhecimentos para aplicação prática, ou seja, pretende-se que a taxonomia resultante possa auxiliar profissionais da TI no processo de engenharia de requisitos.

Desta forma, a seção 2 apresenta algumas classificações resultantes da pesquisa documental e bibliográfica realizada. A seção 3 expõe a taxonomia unificada; finalmente na seção 4 as conclusões e trabalhos futuros são discutidos.

2 TAXONOMIAS DE REQUISITOS NÃO FUNCIONAIS

Na literatura é possível encontrar diversas propostas de taxonomia e organização dos RNFs. A seguir são apresentadas algumas propostas de diferentes autores. A Figura 1 apresenta uma classificação que distingue os requisitos de produto, organizacionais e externos, conforme proposto por Sommerville (2007).

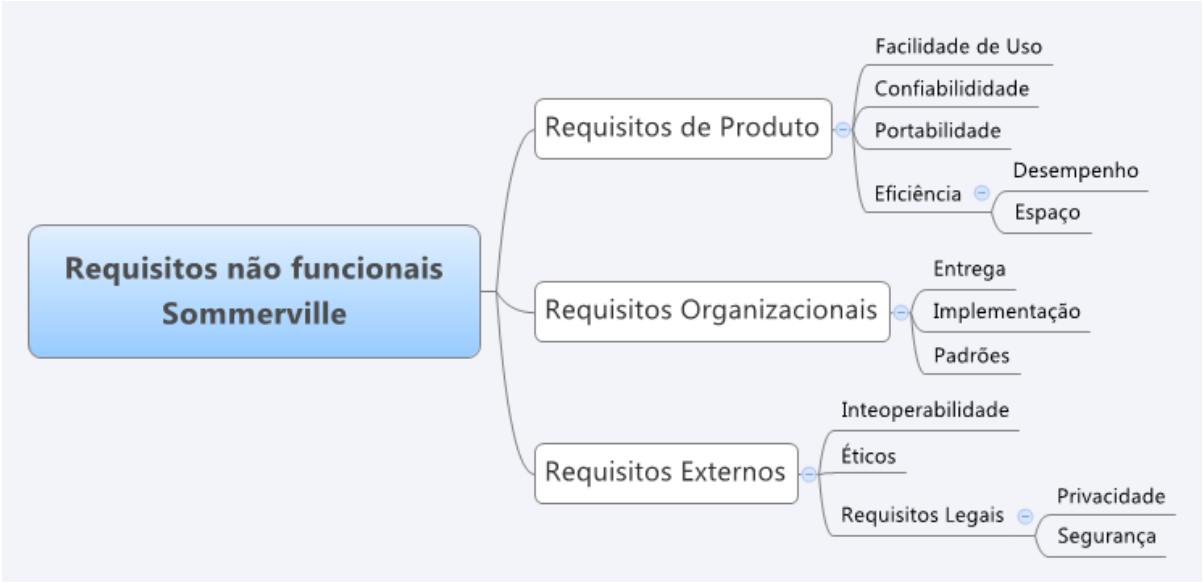

Figura. 1. Taxonomia de RNFs proposta por Sommerville

Fonte: Sommerville (2007)

Outra classificação de RNF que pode ser considerada é a norma ISO/IEC 25010 (2010), que apresenta um modelo de qualidade para produto de software. A norma prevê oito características principais, sendo uma delas, a funcionalidade, associada aos requisitos funcionais e as demais associadas aos RNF. Além disso, cada característica apresentada pela norma está dividida em sub-características (conforme Figura 2).

Figura. 2. Taxonomia de RNFs proposta pela ISO/IEC 25010

Fonte: ISO/IEC 25010 (2010)

O documento da IEEE Std 830-1998 (1998) propõe um conjunto de práticas recomendadas para a especificação de requisitos de software. O documento expõe uma série de características fundamentais que devem ser suficientemente detalhadas para permitir o correto entendimento do sistema. Essas características podem ser facilmente identificadas como RNF. A Figura 3 exibe essas características.

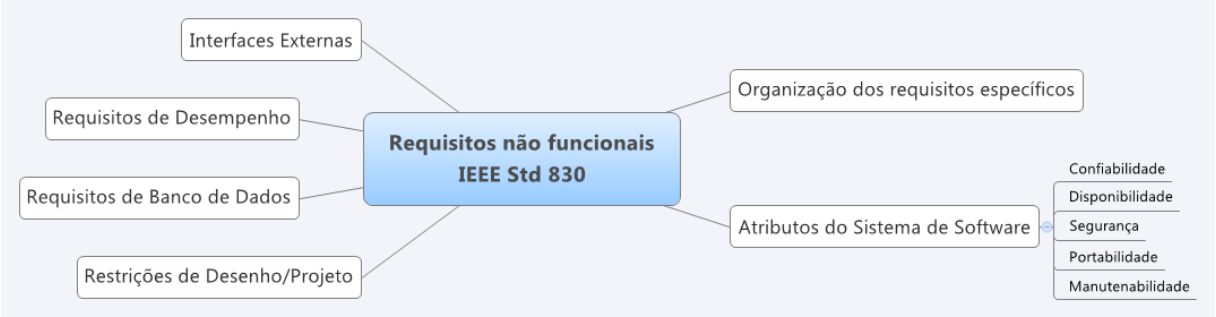

Figura. 3. Taxonomia de RNFs proposta por IEEE Std 830

Fonte: IEEE Std 830 (1998)

O RUP (Processo Unificado da Rational) sugere que uma forma de caracterizar os vários tipos de requisitos é utilizando o modelo FURPS+. O acrônimo FURPS para descrever as principais categorias e subcategorias e o “+” em FURPS+ é para reforçar a inclusão de requisitos como: restrições de projeto, restrições de implementação, requisitos de interface e requisitos físicos (RATIONAL SOFTWARE, 2002). A Figura 4 apresenta a tipologia proposta pelo RUP.

Figura. 4. Taxonomia de RNFs proposta por Rational Software

Fonte: Rational Software (2002)

Conallen (2002), ao abordar requisitos para sistemas web, apresenta cinco categorias de RNF: usabilidade, confiabilidade, segurança, hardware e implantação. O objetivo das categorias, segundo o autor, é torná-los mais fáceis de compreender e controlar. Contudo, Conallen (2002) destaca que a lista não está completa, dependendo da finalidade do sistema, outras categorias também poderiam ser apropriadas.

Withall (2007) apresenta vários padrões de requisitos que foram organizados em oito domínios distintos, cada qual possuindo padrões que objetivam atender funcionalidades e características dos sistemas, aumentando a qualidade no processo de desenvolvimento de software. Para Withall (2007), juntos esses modelos podem representar boa parte do conjunto de especificações típicas de requisitos de um sistema comercial. A Figura 5 apresenta os padrões referentes aos RNF. Os padrões “estrutura de dado”, “identificação”, “fórmula de cálculo”, “taxas”, “entidade ativa”, “transações”, “consulta” e “relatório” foram omitidos, pois não se enquadram como RNF.

Leffingwell (2007) apresenta uma ampla lista com exemplos de RNFs que podem ser aplicados para um determinado contexto do projeto. O autor também sugere utilizar o acrônimo URPS (que significa: usabilidade, confiabilidade, desempenho e suportabilidade) para organizar os requisitos sempre que possível. Esse acrônimo é o mesmo utilizado pelo RUP (RATIONAL SOFTWARE, 2002), excluindo apenas a letra “F” que referencia os requisitos funcionais. O Quadro 1 detalha os RNF propostos por Leffingwell (2007).

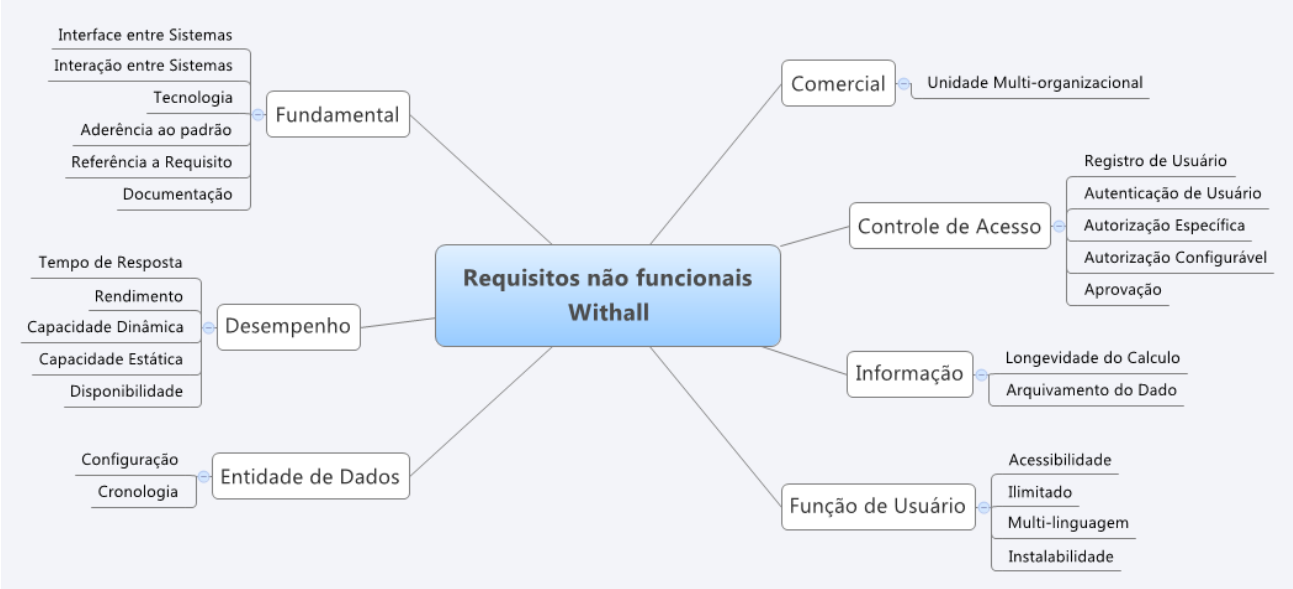

Figura. 5. Taxonomia de RNFs proposta por Withall

Fonte: Withall (2007)

Requisitos não funcionais		
Acessibilidade Auditoria e controle *Backup* Capacidade de operação Certificação Compatibilidade Confiabilidade ▪disponibilidade ▪tempo médio entre falhas ▪precisão ▪numero de defeitos Contrato de custódia	Desempenho ▪tempo de resposta ▪taxa de transferência ▪capacidade atual e futura ▪escalabilidade ▪modos de degradação ▪utilização de recursos Documentação Eficácia Eficiência Estabilidade Extensibilidade Gerencia de configuração Gerenciamento de falhas Interoperabilidade Manutenabilidade	Tecnologia *open source* Portabilidade Preço Privacidade Qualidade Questões legais e de licenciamento Recuperação Recuperação de desastres Resiliência Restrições de recursos Restrições de projeto Robustez Segurança Suportabilidade Testabilidade Usabilidade Violação de patentes

Quadro 1. Exemplos de RNFs

Fonte: Leffingwell (2007)

3 TAXONOMIA UNIFICADA

A taxonomia para RNF proposta nesse trabalho foi elaborada a partir da unificação das diversas classificações apresentadas anteriormente. Essa unificação permitiu criar uma taxonomia mais ampla e diversificada, que abrangesse um maior número de categorias de RNF.

Para melhor organização, a taxonomia está estruturada em dois níveis principais, o primeiro nível foi baseado na taxonomia do Sommerville [1] que divide os RNF em três classes: requisitos de produto que especificam os atributos de qualidade que o sistema deve apresentar; requisitos organizacionais que são derivados de políticas e procedimentos da organização; e requisitos externos que abrangem todos os requisitos derivados de fatores externos ao sistema e o seu processo. No segundo nível, especificamente os requisitos de produto foram divididos conforme a taxonomia proposta pela ISO/IEC 25010 (2010) que apresenta as sete características principais do software: confiabilidade, compatibilidade, usabilidade, eficiência, segurança, manutenibilidade e portabilidade. Os requisitos organizacionais e externos foram divididos segundo taxonomia definida pelo Sommerville (2007). Dessa forma, os requisitos organizacionais foram divididos em requisitos de entrega, requisitos de implementação e requisitos de padrões. Os requisitos externos foram divididos em requisitos de interoperabilidade, requisitos éticos e requisitos legais. A opção por partir da taxonomia de Sommerville (2007) considerou a maior abrangência dos primeiros níveis (em relação às outras classificações propostas). Já a escolha da ISO/IEC 25010 (2010) deve-se, tanto à abrangência da classificação, quanto à disseminação das normas ISO junto à comunidade, podendo ser este um fator facilitador da adoção desta taxonomia unificada.

As demais classificações foram organizadas dentro desses níveis, gerando o Quadro 2 com os RNF de produto, o Quadro 3 com os RNF organizacionais e o Quadro 4 listando os RNF externos. Visando a manter a "rastreabilidade" com a origem de cada classificação proposta é utilizada a seguinte legenda: [1] Sommerville (2007); [2] Conallen (2002); [3] ISO/IEC 25010 (2010); [4] IEEE (1998); [5] RUP (RATIONAL SOFTWARE, 2002); [6] Withall (2007); e [7] Leffingwell (2007). Visando a facilitar o entendimento, uma breve descrição sobre cada classificação foi incluída, bem como, para algumas situações, um exemplo é apresentado.

Requisitos de produto [1]
Confiabilidade [1,2,3,4,5,7] – capacidade do sistema realizar e manter seu funcionamento em circunstâncias de rotina, bem como em circunstâncias hostis e inesperadas 1. **Disponibilidade** [3,4,5,6,7] – período de tempo que o software deverá estar disponível para os usuários. i) **Recuperabilidade** [3,5,7] – período de tempo necessário para retornar ao funcionamento após uma falha. ii) **Recuperação de desastres** [7] - políticas e procedimentos para recuperação de desastres naturais ou induzidos pelo homem. 2. **Eficácia** [7] – capacidade de executar perfeitamente uma tarefa conforme previamente planejado. Exemplo: o sistema deverá apresentar uma eficácia superior a 998 para cada 1000 saques em caixas 24 horas.

3. **Exatidão/precisão** [5,7] - grau de exatidão ou precisão no processamento e apresentação dos resultados.
4. **Número de defeitos** [7] – referencia o número máximo de defeitos por unidade. Exemplo: máximo de 20 defeitos por cada mil linhas de código (KLOC)
5. **Maturidade** [3] – grau de maturidade indica o quão sólido o software é. Pode ser avaliado principalmente pelo número de defeitos, que deve estar em declínio com o amadurecimento do software.
6. **Previsão de confiabilidade** [5] – capacidade de estimar a confiabilidade ou o número de defeitos latentes do produto quando ele estiver disponível para os clientes. Exemplo: em um sistema médico, a consulta de resultados de exame precisa estar funcional 97% do tempo.
7. **Resiliência/robustez/tolerância a falhas** [3,7] – capacidade de o sistema funcionar mesmo em condições anormais. Exemplo: o sistema médico deve permitir a consulta de prontuários locais quando perder a comunicação com o sistema central, sinalizando a situação anormal.
8. **Tempo médio entre falhas** [5,7] – Tempo médio entre falhas encontradas.
9. **Frequência e gravidade de falha** [5] – Frequência e gravidade de falhas encontradas.

Compatibilidade [3,5,7] – Capacidade do produto de software, sistema ou componente trocar informações com outros produtos, sistemas ou componentes e/ou realizar suas funções necessárias, mesmo compartilhando o mesmo hardware, software ou ambiente.

- Compatibilidade entre softwares, ferramentas e normas [7]
- Compatibilidade entre plataformas [7]
- Compatibilidade com ambientes operacionais [7]

1. **Coexistência** [3] - Capacidade de um produto de software coexistir com outros produtos em ambiente comum;
2. **Interoperabilidade** [1,3, 7] – define como o sistema interage com sistemas em outras organizações.
   i) **Interface e interações entre sistemas** [4, 5,6] – especifica detalhes sobre a interface entre diferentes sistemas ou componentes externos.
   - Restrições de formatos, tempos ou outros fatores usados por tal interação [5] – especifica como o sistema deve interagir, quais as restrições de formatos e outros fatores para essa interação. Exemplo: Para consultar o nome da rua por meio do CEP, deve ser utilizada a interface CEPBusca enviando o CEP no formato #####-###.

Performance e eficiência [1,3,5,7] –utilização adequada dos recursos de forma a maximizar os resultados pré-determinados. Exemplo: o relatório é finalizado em tempo menor do que o esperado.

1. **Capacidade atual e futura (escalabilidade)** [6,7] – definição da capacidade atual e formas de crescimento para o sistema. Exemplo: o cadastro de clientes deve comportar um crescimento de cem mil clientes por ano.
   i) **Capacidade dinâmica** [6] – capacidade de processamento simultâneo do sistema.
   ii) **Capacidade estática** [6] – capacidade de armazenamento do sistema (BD)
2. **Rendimento / velocidade / taxa de transferência** [5,6,7] – índice ou taxa na qual o sistema deve ser capaz de executar como, por exemplo, 10 inscrições por minuto.
3. **Desempenho (tempo de resposta)** [1,3,4,5,6,7] - especificar o tempo de resposta para uma solicitação.
4. **Acurácia** [3] - Capacidade do produto de software atender às exigências dos limites máximos de um parâmetro de produto ou sistema.
5. **Comportamento em relação aos recursos** [3] - Capacidade do produto de software de usar a quantidade de recursos apropriada

Manutenibilidade/suporte [3,4,5,7] – capacidade ou facilidade do produto de software ser modificado, incluindo tanto melhorias quanto correções de defeitos e falhas.

1. **Modificabilidade/estabilidade** [3,7] – capacidade do software de evitar efeitos colaterais decorrentes de modificações introduzidas. Exemplo: número de erros decorrentes de modificações inferior a 5 para cada mil linhas de código (KLOC)
2. **Analisabilidade** [3] – Esforço necessário para diagnosticar deficiência ou causa das falhas e identificar as partes do software a serem modificadas.
3. **Extensibilidade** [5,6,7] - esforço necessário para modificar um software, seja removendo erros ou melhorando seu desempenho. Exemplo: o sistema deve permitir adicionar extensões (bibliotecas) sem a necessidade de reiniciar.
4. **Gerenciamento de falhas** [7] – indica a necessidade de gerenciar falhas, como por exemplo, registrar no *log* as falhas ocorridas no sistema.
5. **Gerência de configuração** [5,6,7] - indica a necessidade de controle contínuo das mudanças. Compreende a identificação da configuração, o controle das mudanças efetuadas e a rastreabilidade.
6. **Possibilidade de serviço** [5] – possibilidade de executar determinadas operações na forma de serviços. Exemplo: o sistema deve possuir um serviço para atualizações automáticas de versões.
7. **Testabilidade** [3,5,7] – facilidade em executar testes e encontrar problemas no software. Exemplo: o sistema deve possuir 80% de cobertura nos testes automatizados.

8. **Modularidade** [3] - Capacidade do produto de software ser composto de componentes modulares de forma que uma mudança em um componente tenha impacto mínimo nos outros componentes;
9. **Reusabilidade** [3] - Capacidade de um módulo ser utilizado em mais de um produto de software, ou ser utilizado como base para se desenvolver outros módulos.

Portabilidade [1,3,4,7] – facilidade de transpor um software de um ambiente a outro.

1. **Adaptabilidade [3,5]** – capacidade do software se adaptar a diferentes ambientes sem a necessidade de ações adicionais (configurações). Exemplo: o sistema deve funcionar em sistemas 32 bits e 64 bits.
2. **Facilidade de substituição** [3] - capacidade e esforço necessário para substituir outro software ou componente, com o mesmo propósito e mesmo ambiente. Exemplo: as versões disponíveis para o sistema devem ser compatíveis com a versão original.
3. **Facilidade de instalação** [3,5,6] – grau de facilidade para instalar e atualizar o sistema ou parte dele.

Usabilidade [1,2,3,5,7] – grau de facilidade de utilização do software.

1. **Acessibilidade** [3, 6,7] – extensão em que um sistema ou parte dele deve ser acessível por pessoas com certo tipo de deficiência ou outra necessidade específica.
2. **Ajuda *on-line* e contextual** [5] – indica a necessidade de ajuda *on-line* e contextual.
3. **Assistentes e agentes** [5] – indica a necessidade de assistentes e agentes.
4. **Consistência na interface do usuário** [5] – diretrizes gerais aplicáveis à criação da interface do usuário. Exemplo: a interface do usuário deve ser elaborada utilizando o conceito de janelas e menus gráficos.
5. **Estética/fatores humanos** [3, 5] – diretrizes gerais aplicáveis na aparência do software. Exemplo: utilizar fonte Ariel tamanho 12 na cor preta com fundo claro.
6. **Apreensibilidade** [3] – esforço necessário para aprender a utilizar as potencialidades oferecidas pelo sistema. Exemplo: em um jogo infantil, haver uma apresentação inicial em que os comandos básicos são apresentados por um personagem.
7. **Inteligibilidade** [3] – indica a facilidade com que o usuário pode compreender as funcionalidades do software e avaliar se elas podem ser usadas para satisfazer suas necessidades. Exemplo: o sistema deve exibir *tooltips* sobre os menus e botões com uma breve descrição da funcionalidade.
8. **Manual do usuário** [5] – indica a necessidade de elaboração de manual do usuário.

9. Materiais de Treinamento [5] – indica a necessidade de materiais de treinamento.

10. Múltiplo (suportar múltiplas empresas e moedas) [6] – especifica que o software deve acomodar múltiplas empresas e moedas ao mesmo tempo.
11. Internacionalização [5,6] – especifica que um sistema deve apresentar sua interface em mais de uma língua.
12. Operabilidade/operacionalidade [3] - atributos do software que evidenciam o esforço do usuário para sua operação e controle. Exemplo: em um sistema para biblioteca, o empréstimo e devolução de um livro devem ser realizados em uma única janela/tela.
13. Proteção contra erros [3] - capacidade do sistema proteger o usuário de cometer erros.

Segurança [1,2,3,4,6,7] – especifica os requisitos de segurança
1. Auditoria e Controle [7] – especificam os aspectos a serem contemplados para viabilizar a auditoria e controle.
2. Integridade [3] - capacidade do produto de software prevenir acessos ou modificações de programas de computador ou dados;
3. Contestabilidade e responsabilização [3] - quantidade de ações ou eventos que podem ter sua ocorrência comprovada, de modo que os eventos ou ações não possam ser contestados mais tarde, bem como registro de ações de uma entidade podendo identificar exclusivamente as ações à uma entidade.
   i) Cronologia (*logs*) [6] – especifica a utilização de logs para registrar alterações no software.
4. Autenticidade [3] - capacidade do produto de software identificar que um objeto ou recurso é realmente quem ele declara ser.
5. Confidencialidade/controle de acesso [3,6] – capacidade do produto de software de garantir que os dados estarão acessíveis somente aos usuários que possuem autorização de acesso;
   - Registro de usuário [6] – especifica como novos usuários são registrados.
   - Autenticação de usuário [6] – especifica que uma pessoa deve ser autenticada para acessar áreas não públicas.
   - Autorização específica [6] – especifica que um conjunto de usuários está autorizado ou não a fazer ou ver coisas.
   - Autorização configurável [6] – especifica que a autorização do usuário é configurável.
   - Aprovação [6] – especifica que uma determinada ação deve ser aprovada.

Quadro 2. RNF de produto

Fonte: elaborado pela própria autora

Requisitos organizacionais [1]

Restrições de desenho/projeto [5,4,7]

1. **Implantação** [2] – indica como o sistema deverá ser instalado e acessado pela equipe de atualização.
2. **Implementação** [1,5] – especifica ou restringe a construção de um sistema. Como, por exemplo, a especificação da linguagem de implementação, da arquitetura, do banco de dados, dos ambientes operacionais.
3. ***Backup*** [7] – especifica as regras para criação dos *backups.*
4. **Banco de dados** [4] – indica as restrições de banco de dados.
   - **Arquivamento de dados** [6] – especifica a movimentação ou cópia dos dados de um local para outro.
   - **Longevidade do dado** [6] – especifica quanto tempo um dado deve ser mantido ou estar disponível para acesso.
   - **Políticas de integridade de banco de dados** [5] – restrições sobre políticas de integridade de banco de dados.
5. **Fundamental** [6]
   - **Documentação** [6,7] – especifica o formato e abrangência da documentação.
   - **Referência a requisito** [6] – especificar quais requisitos externos devem ser atendidos como requisitos presentes no projeto.
   - **Tecnologia** [6, 7] – restringe a utilização de determinada tecnologia.
   - **Linguagens de implementação [5]** – restringe a utilização de determinada linguagem de implementação.
   - **Padrões** [1,5,6] – especifica a utilização de determinado padrão.
6. **Hardware [2]** – requisitos de hardware do sistema.
7. **Organização dos requisitos específicos** [4] – requisitos específicos tendem a ser extensivas quando se trata de um sistema não trivial, por isso os requisitos devem ser organizados de uma forma simples de compreender. Exemplo: organização por módulos.
8. **Requisito físico [5]** – requisitos físicos do produto. Exemplo: deve ser construído com material resistente a pequenas quedas (até 1 metro), com tamanho não ultrapassando 10x15 cm e peso máximo de 300 gramas.
   - Material [5] – restrições quanto aos materiais utilizados no produto.
   - Forma [5] – restrições quanto ao formato do produto.
   - Tamanho [5] – restrições quanto ao tamanho do produto.
   - Peso [5] – restrições quanto ao peso do produto.
9. **Restrições de recursos** [5,7] – especificam as restrições de recursos como, por exemplo, velocidade do processador, memória, espaço em disco, largura de banda de rede, entre outros.

Requisitos comerciais [6]

1. **Entrega** [1] – especificam quando o produto e seus documentos devem ser entregues.

2. **Preço** [7] – especificam limites quanto ao preço do produto.
3. **Unidade multi-organizacional** [6] – especifica um tipo de estrutura organizacional que o sistema deve suportar. Exemplo: o sistema deve ser capaz de suportar múltiplas empresas simultaneamente. Cada empresa terá seus próprios empregados que utilizam o sistema. Uma instalação do sistema deverá acomodar até uma dúzia de empresas.

Quadro 3. RNF organizacionais

Fonte: elaborado pela própria autora

Requisitos externos [1]
Requisitos legais [1] – devem ser seguidos para assegurar que o sistema opere de acordo com a lei. 1. **Certificação** [7] – especifica as certificações ou padrões mínimos exigidos para o funcionamento do sistema. Exemplo: o sistema deve apresentar o certificado "*site* blindado". 2. **Contrato de custódia** [7] – estabelece restrições relativas à custódia do software. 3. **Privacidade** [1,7] – estabelece restrições relativas à privacidade do software. 4. **Questões legais e de licenciamento** [7] – contrato de licença de usuário final ou acordo de licença de software entre o concedente e o comprador, estabelecendo o direito do comprador de utilizar o software. 5. **Ilimitado** (instalar em outras empresas/filiais ou vendido) [6] – definir que o sistema poderá ser instalado em outras empresas/filiais ou vendido. 6. **Violação de patentes** [7] – estabelece a necessidade de controlar a violação de patentes.
Requisitos éticos [1] – requisitos definidos para garantir a aceitação do software pelos seus usuários e o público em geral. Exemplo: em um sistema médico, o sistema não deve apresentar aos usuários quaisquer dados de cunho privativo.

Quadro 4. RNF externos

Fonte: elaborado pela própria autora

Esta taxonomia buscou reunir todos os RNF abordados pelos diferentes autores, agregando um total de noventa aspectos não funcionais distintos. Destes aspectos, sessenta estão relacionados com atributos de qualidade/ restrições ao produto, 23 foram considerados aspectos organizacionais (sendo vinte de desenho/projeto) e sete relacionados a requisitos externos. A Figura 6 apresenta a sumarização da quantidade de aspectos considerados em cada nível da taxonomia proposta.

Figura. 6. Sumarização da taxonomia unificada

Fonte: elaborado pela própria autora

Por fim é importante destacar dois pontos que facilitam o entendimento da taxonomia: (i) a rastreabilidade de sua origem está claramente identificada (permitindo a consulta à fonte e acesso a maior detalhamento); e (ii) cada RNF possui uma descrição visando a facilitar o entendimento. Além disso, diversos requisitos foram exemplificados para auxiliar na compreensão.

4 CONSIDERAÇÕES FINAIS

Este artigo apresentou um estudo sobre RNF e sobre as diferentes taxonomias de RNF disponíveis na literatura, envolvendo oito classificações distintas. Posteriormente, foi proposta uma taxonomia criada a partir da unificação das classificações existentes, buscando contemplar uma variedade maior de requisitos não funcionais.

Entende-se que a taxonomia unificada pode auxiliar o processo de desenvolvimento de software nos seguintes aspectos: (i) ser utilizada como um guia durante a fase de elicitação de requisitos, evidenciando a necessidade de análise de algum RNF importante; (ii) permite a especificação de RNF de forma mais padronizada; (iii) o fato dos requisitos estarem organizados em uma taxonomia padrão facilita a reutilização de requisitos; e (iv) a utilização de uma taxonomia pode auxiliar na verificação e validação dos requisitos.

Destaca-se que esta taxonomia consolidou apenas 8 fontes distintas, as quais são importantes referenciais teóricos da área, bem como literatura bem consolidada e conhecida. Contudo, reconhece-se ser esta uma primeira versão de uma taxonomia unificada. Um mapeamento sistemático da literatura pode ampliar os aspectos contemplados pela taxonomia proposta, sendo este um próximo passo no estudo.

Na sequência, a elaboração de padrões de requisitos (no estilo proposto por Withall (2007)) para todos os aspectos incorporados na taxonomia e a disponibilização de um mecanismo de busca e apresentação devem

facilitar a adoção da taxonomia. Por fim, prevê-se a avaliação/ampliação da taxonomia pela comunidade (por meio de um *survey*) e a aplicação em projetos e estudos de caso.

REFERÊNCIAS

BASTOS, F. A. M. F. Elicitação de requisitos não funcionais em conformidade com políticas de qualidade para aplicações médicas. Tese de doutorado submetida à UFMA, 2007.

CHICHINELLI, M. Contribuição da técnica de modelagem organizacional ao processo de engenharia de requisitos, com destaque aos requisitos não funcionais. Tese de doutorado submetida à USP. São Paulo, 2002.

CHUNG, L.; LEITE, J. C. S. P. On non-functional requirements in software engineering. In: Conceptual Modeling: Foundations and Applications. BORGIDA, A. T.; CHAUDHRI, V. K.; GIORGINI, P.; YU, E. S. (eds.). *Lecture notes in Computer Science*, v. 5600, p. 363-379. Springer-Verlag, 2000. http://dx.doi.org/10.1007/978-3-642-02463-4_19

CONALLEN, J. Building web applications with UML, 2. ed. Boston: Pearson Education, 2002.

CYSNEIROS, L. Requisitos não funcionais: da elicitação ao modelo conceitual. Tese de doutorado submetida à PUC/RJ. Rio de Janeiro, 2001.

GIL, A. C. Como elaborar projetos de pesquisa. 4. ed. São Paulo: Atlas, 2007.

IEEE - INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. IEEE recommended practice for software requirements specifications, Std 830-1998. USA, 1998.

ISO/IEC 25010:2010. Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) — System and software quality models. Switzerland, 2010.

LEFFINGWELL, D. Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Boston: Addison Wesley, 2007.

RATIONAL SOFTWARE. Rational Unified Process. IBM, 2002.

SOMMERVILLE, I. Engenharia de software. 8. ed. São Paulo: Addison Wesley, 2007.

WITHALL, S. Software requirement patterns. Washington: Microsoft, 2007.