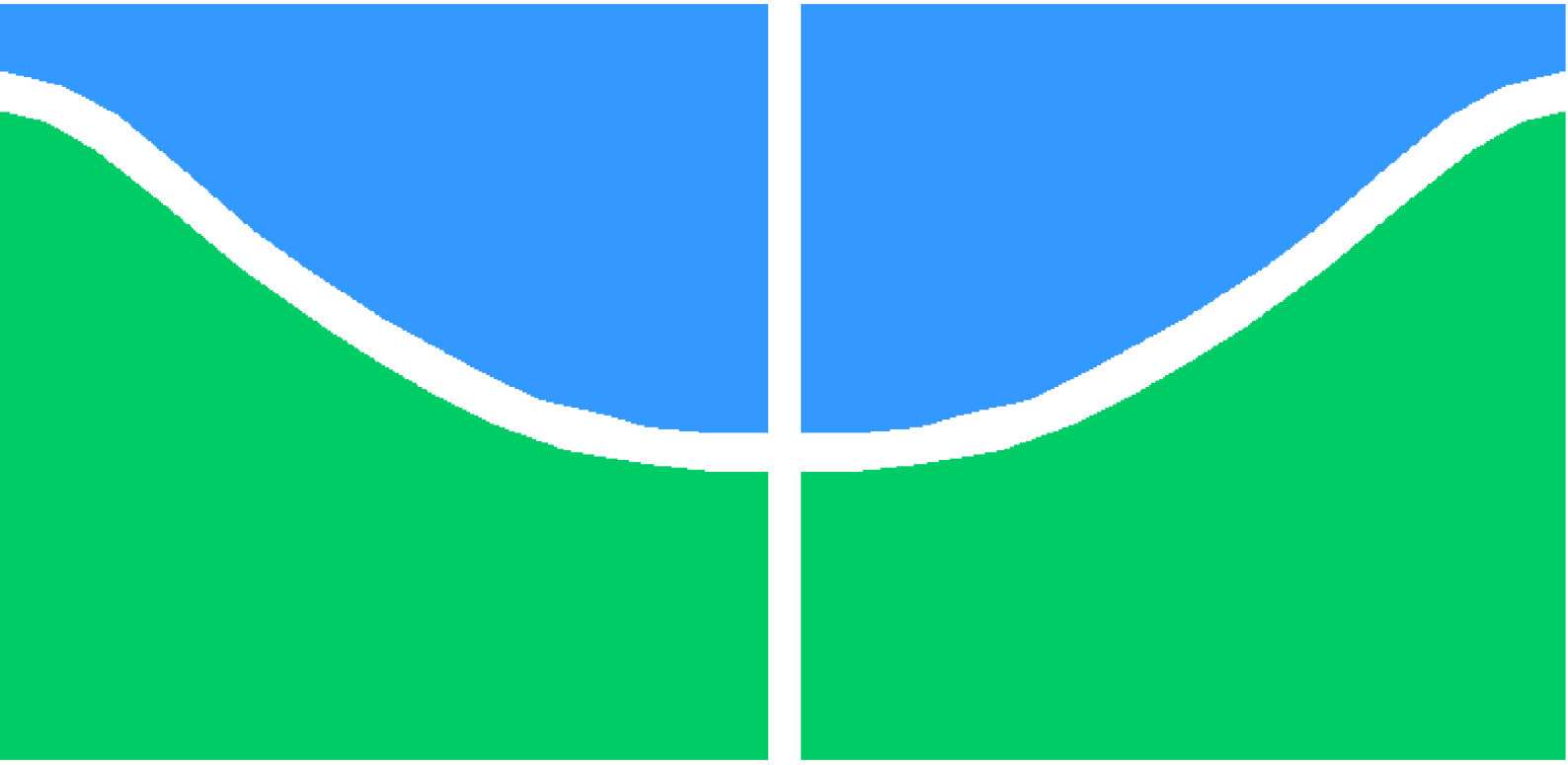


Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Catálogo de Segurança para o Padrão Arquitetural MVC: Modelagem Orientada a Grafos de Interdependências de Requisitos Não Funcionais

Autor: Ebenezer Andrade da Silva

Orientadora: Profª Dra. Milene Serrano

Coorientador: Profº Dr. Maurício Serrano

Brasília, DF

2019

Ebenezer Andrade da Silva

Catálogo de Segurança para o Padrão Arquitetural MVC: Modelagem Orientada a Grafos de Interdependências de Requisitos Não Funcionais

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientadora: Prof[a] Dra. Milene Serrano

Coorientador: Prof[o] Dr. Maurício Serrano

Brasília, DF

2019

Ebenezer Andrade da Silva

Catálogo de Segurança para o Padrão Arquitetural MVC: Modelagem Orientada a Grafos de Interdependências de Requisitos Não Funcionais/ Ebenezer Andrade da Silva. – Brasília, DF, 2019-

87 p. : il. (algumas color.) ; 30 cm.

Orientadora: Profª Dra. Milene Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2019.

1. NFR Framework. 2. Catálogo de Segurança. I. Profª Dra. Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Catálogo de Segurança para o Padrão Arquitetural MVC: Modelagem Orientada a Grafos de Interdependências de Requisitos Não Funcionais

CDU 02:141:005.6

Ebenezer Andrade da Silva

Catálogo de Segurança para o Padrão Arquitetural MVC: Modelagem Orientada a Grafos de Interdependências de Requisitos Não Funcionais

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Profª Dra. Milene Serrano
Orientadora

Profº Dr. Maurício Serrano
Coorientador

Profº Dr. André Barros de Sales
Convidado 1

Brasília, DF
2019

Esta monografia é dedicada aos meus pais, Elenira Andrade e José Elieser Filho, que me educaram, me ensinaram, e suportaram a distância e a saudade de um filho, para que esta monografia pudesse ser escrita. Em homenagem, dedico ainda essa monografia à memória de minha querida avó paterna, Maria Alicina da Silva.

Agradecimentos

Agradeço, primeiramente, a Deus, que sempre foi meu porto seguro e me deu forças para enfrentar e superar todas as dificuldades.

Agradeço à excelente Universidade de Brasília e ao corpo docente de graduação em Engenharia de Software, em especial aos professores, Profª. Milene Serrano e Profº. Maurício Serrano, por toda paciência e disponibilidade em me orientar; pelo suporte; pelas suas correções e incentivos, e por ensinarem conceitos de extrema importância nas áreas de Desenho de Software e Requisitos de Software; e ao Profº. George Marsicano Corrêa, por ter aberto meu horizonte sobre Requisitos de Software de uma maneira tão fantástica e exemplar.

Aos meus pais, meus avós, meus irmãos, bem como ao meu cunhado, Flávio Carlos, por todo amor, incentivo e apoio durante toda minha graduação.

Aos irmãos na amizade, Jonas da Silva e Heleno da Silva, por confiarem em meu potencial, incentivando-me sempre.

Ao meu primo, Oziel da Silva, por me auxiliar nos momentos difíceis.

A meus amigos de graduação, Matheus Silva, Pedro Ivo de Andrade, Attany Araújo, Keli Cristina, Vinicius Bandeira, Maxwell Oliveira, Thiago Nunes, Yan Watanabe,Tainara Reis, João Paulo Mendonça, Hebert Douglas, Renan Costa, João Henrique, Bruno Contessotto, Paulo Markes e Vinicius Pinheiro por toda ajuda com dúvidas e troca de conhecimento durante minha graduação.

Aos apoiadores, Adriana Matrins, Vick Martins, Omar Júnior, Omar Faria, Ygor Mota, Laisa Paiva, Maria Walcira, Franscico de Sousa, Rafael Paiva, Cirilo Júnior, Victor Hugo, Gabriela Nara, Luiza Lopes, Valter Avelino, Yasmin Nabil, Luisa de Oliveira, Eliana e Odair, por me acolherem nos dias difíceis longe da minha família, e me darem apoio emocional durante minha graduação e à escrita desse trabalho.

À Milena Lima, por ser conselheira e por todo apoio emocional para que eu pudesse redigir este trabalho, acreditando em meu potencial e incentivando-me.

A todos que de forma direta ou indireta contribuiram para realização deste trabalho, e fizeram parte da minha formação.

Meu muito obrigado!

"Somente pensamos quando confrontados com um problema."

John Dewey

Resumo

Este trabalho teve como principal objetivo a construção de um catálogo de segurança para o Padrão Arquitetural MVC (*Model-View-Controller*), visando auxiliar Engenheiros de Requisitos e Engenheiros de Software na especificação dos requisitos associados ao requisito não funcional Segurança. O catálogo encontra-se documentado com base em uma modelagem orientada a grafos de interdependência de requisitos não funcionais, levando em consideração boas práticas da Engenharia de Requisitos Orientada à Meta. Na construção do catálogo, foram considerados alguns pilares, sendo esses: conceitos associados à Segurança e fundamentados em autores da área, com ênfase em Confidencialidade, Integridade e Disponibilidade, e a definição de Segurança da Informação estabelecida pela ISO 27001. Como uma extensão das contribuições base do catálogo, os apontamentos do mesmo foram mapeados/associados às camadas do Padrão MVC, no intuito de auxiliar ainda mais os usuários do catálogo, enquanto desenvolvem suas aplicações baseadas nesse padrão arquitetural em três camadas. Por fim, tem-se a aplicação do catálogo em diferentes cenários de uso, o que possibilitou não apenas um catálogo em alto nível de abstração, baseado nas interdependências dos requisitos não funcionais correlacionados à Segurança, mas também um catálogo que acorda uma série de operacionalizações, as quais procuram cumprir (de forma satisfatória pelo menos) com os requisitos não funcionais especificados.

Palavras-chaves: Catálogo de Segurança de Software. Padrão Arquitetural *Model-View-Controller*. Engenharia de Requisitos Orientada à Meta. Segurança. Confidencialidade. Integridade. Disponibilidade.

Abstract

The main objective of this research was the construction of a Security Catalog for the MVC (Model-View-Controller) Architectural Pattern, aiming at assisting Requirements Engineers and Software Engineers in specifying the requirements associated with Security, a non-functional requirement. The catalog is documented based on a graph-oriented modeling of interdependence of non-functional requirements, taking into account good practices of Goal-Oriented Requirements Engineering. In the construction of this catalog, some pillars were considered, such as concepts associated to Security and substantiated on authors of the area, with emphasis on Confidentiality, Integrity and Availability, in addition to the definition of Information Security established by ISO 27001. As an extension of the catalog's base contributions, it has been mapped to/associated with the MVC pattern layers in order to further assist catalog users while developing their applications based on this three-tier Architectural Pattern. Lastly, the catalog has been applied in different usage scenarios, which has made possible not only a catalog with a high level of abstraction, based on the interdependencies of the non-functional requirements correlated with Security, but also a catalog that agrees a series of operations, which they seek to fulfill (at least satisfactory) with the specified non-functional requirements.

Key-words: Software Security Catalog. Architectural Pattern Model-View-Controller. Goal-Oriented Requirements Engineering. Security. Confidentiality. Integrity. Availability.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
BPMN	*Business Process Model and Notation*
CIA	*Confidenciality, Integrity, Availability*
CSS	*Cascading Style Sheets*
DFD	*Data Flow Diagram*
ERD	*Entity Relationship Diagram*
FURPS	*Functionality, Usability, Reliability, Performance, Supportability*
GORE	*Goal-Oriented Requirements Engineering*
GPL	*General Public License*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
IEC	*International Electrotechnical Commission*
ISO	*International Organization of Standardization*
KAOS	*Knowledge Acquisition in autOmated Specification*
MDA	*Model Driven Architecture*
MTBF	*Mean Time Between Failure*
MVC	*Model-View-Controller*
NBR	Norma Brasileira
QR	*Quick Response*
RE-Tools	*Requirements Engineering - Tools*
RNF	Requisito Não Funcional
RoR	*Ruby on Rails*
SGBD	Sistema de Gerenciamento de Banco de Dados

SIGs	*Softgoal Interdependecy Graphs*
TCC	Trabalho de Conclusão de Curso
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*

Sumário

1 Introdução

Neste capítulo, são descritos o contexto, no qual esse trabalho está inserido; a questão de pesquisa, a qual se procurou responder com a realização do trabalho; a justificativa, visando destacar a contribuição dessa pesquisa; os objetivos geral e específicos, os quais foram alcançados com a realização do trabalho, e por fim, a organização dos demais capítulos desse trabalho.

1.1 Contextualização

Os requisitos não-funcionais (RNFs) e funcionais descrevem as características de um software, focando nas questões: **como esse software deve fazer?** e **o que esse software deve fazer?** (SOMMERVILLE; SAWYER, 1997). Frequentemente, os RNFs são especificados de forma equivocada ou ainda são negligenciados, não sendo especificados (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016). Visando auxiliar os Engenheiros de Requisitos na tarefa de especificar os RNFs, os autores, em (MAIRIZA; ZOWGHI; NURMULIANI, 2010), destacam 114 classes de RNFs. Dentre essas classes, as mais mencionadas na literatura são: 1º Desempenho, 2º Confiabilidade e 3º Usabilidade. No contexto empresarial, ou seja, mais aplicado e comercial, as classes de RNFs mais utilizadas nas especificações são: 1º Segurança, 2º Confiabilidade e 3º Usabilidade (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016). Como o foco desse trabalho é atuar especificamente em suportes aplicáveis no contexto empresarial, o foco de contribuição desse trabalho concentra-se na especificação do RNF de segurança, dado que é a classe de RNF mais relevante para o mercado, de acordo com a literatura investigada.

Ao desenvolver um software, os RNFs impactam, segundo (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016), na implementação, manutenção, operacionalização e utilização de recursos. Adicionalmente, impactam em aspectos arquiteturais de uma aplicação de software, sendo necessário especificar a arquitetura orientando-se não apenas pelos requisitos funcionais, mas também pelos RNFs (BUSCHMANN et al., 1996).

Dependendo do padrão arquitetural utilizado, as especificações bem como as operacionalizações dos RNFs podem variar (CHUNG et al., 2012). Diante do exposto, e visando focar em um padrão arquitetural comumente utilizado, o presente trabalho orientar-se pelo padrão arquitetural *Model-View-Controller* (MVC). Trata-se de um padrão bem aceito, bem como utilizado no desenvolvimento de aplicações Web, sendo inclusive base para o desenvolvimento dessas aplicações em *frameworks* e plataformas de geração de código mais emergentes, orientadas à Convenção sobre Configuração (JAILIA et al., 2016).

O MVC é um padrão arquitetural para sistemas interativos, que divide a aplicação em componentes: a *Model*, que contém a implementação das funcionalidades principais e os dados da aplicação, ou seja as entidades de domínio; a *View*, que exibe as informações da aplicação ao usuário, sendo, portanto, a camada mais próxima desse último, e a *Controller*, que lida com as entradas do usuário (BUSCHMANN et al., 1996), sendo um componente intermediário entre *Model* e *View*.

Existem alguns *frameworks* conceituais centrados na especificação de RNFs. Dentre eles, destacam-se: NFR *Framework* (CHUNG; LEITE, 2009), i* (HORKOFF; YU, 2006) e FURPS (UMAR; KHAN, 2011). Detalhes sobre esses *frameworks* serão cobertos no Capítulo 2 desse trabalho, Referencial Teórico. Em um primeiro momento, pode-se destacar que os dois primeiros, NFR *Framework* e i*, fazem uso de uma abordagem mais emergente para especificação desses RNFs. Já o FURPS pode ser entendido como um documento simples, especificado em linguagem natural, que se orienta por alguns RNFs comumente encontrados na literatura, no caso (em inglês): *Functionality*, *Usability*, *Reliability*, *Performance* e *Supportability*.

A abordagem mais emergente, mencionada para os *frameworks* NFR e i*, usa modelos específicos, focados na especificação de requisitos usando princípios da *Goal-Oriented Requirements Engineering* (GORE) (HORKOFF et al., 2016). Rastreabilidade de requisitos (WIEGERS; BEATTY, 2013); especificação de impactos e interdependências entre os requisitos; registro de alternativas quanto aos requisitos analisados, e definição de operacionalizações para viabilizar a realização dos requisitos são algumas vantagens que ficam evidentes no uso desses modelos mais específicos, como, por exemplo: o *Softgoal Interdependency Graphs* (SIGs) (CHUNG et al., 2012) ou, em português, Grafos de Interdependências entre Requisitos Não Funcionais.

Portanto, esse trabalho propôs a definição de um Catálogo de SIGs centrado no RNF Segurança, especificamente desenhado para o Padrão Arquitetural MVC, usando a notação do NFR *Framework*. Dentre as contribuições, procura-se auxiliar na tarefa de especificação de requisitos, quando uma arquitetura é imposta. Como o catálogo ficaria muito abrangente e generalista, caso fosse definido para qualquer RNF e padrão arquitetural, não sendo, portanto, útil, focou-se: (i) no RNF mais preocupante - Segurança, segundo a literatura - em aplicações de software de cunho comercial; (ii) em aplicações Web, e (iii) no padrão arquitetural MVC. Vale ressaltar ainda que os termos RNFs, atributos de qualidade, critérios de qualidade e metas flexíveis serão utilizados como sinônimos ao longo desse trabalho.

1.2 Questão de Pesquisa

Este trabalho procurou responder a seguinte questão de pesquisa: *Como auxiliar Engenheiros de Requisitos e Engenheiros de Software na especificação de RNFs quando uma arquitetura é imposta?* É relevante comentar que as considerações a respeito desse questionamento, considerando os resultados obtidos nesse trabalho, parecerão - ao leitor - uma resposta parcial e não plenamente concretizada. Tem-se o cuidado de fazer essa observação, desde já, pois a resposta será focada em um contexto de desenvolvimento de software e não no desenvolvimento de software abrangente e aderente a qualquer necessidade computacional. No caso, como já comentado anteriormente, o trabalho está focado em: um requisito não funcional chave para aplicações comerciais, no caso, Segurança de Software; um tipo de aplicação, sendo esse Web, e no Padrão Arquitetural MVC.

Entretanto, é também relevante considerar que esse contexto de desenvolvimento atende: (i) ao RNF destacado como a principal preocupação em cenários empresariais, com base em autores da área; (ii) ao tipo de aplicação mais comumente desenvolvida na área, e (iii) ao Padrão Arquitetural mais recomendado para esse tipo de aplicação. Portanto, o que parecia uma resposta parcial, por se tratar de resultados endereçados a um contexto de desenvolvimento específico, na verdade, corresponde a uma resposta mais abrangente, dadas as considerações acordadas. Adicionalmente, esse trabalho procura revelar detalhes da construção, aplicação e evolução do catálogo. A intenção é estimular novas linhas de pesquisa, considerando outros RNFs chave bem como outros estilos arquiteturais, cabendo ao interessado usar o catálogo proposto como base para novas iniciativas.

É recomendado que uma questão de pesquisa seja organizada em três visões (KEELE, 2007): **População**, estabelecendo quem são as pessoas diretamente afetadas pela intervenção; **Intervenção**, comparando dois ou mais "tratamentos alternativos", ou seja, duas ou mais possíveis estratégias de atuação, e **Saídas**, estabelecendo a forma como a intervenção vem sendo aplicada. Nesse contexto, tem-se:

- População: "*..Engenheiros de Requisitos e Engenheiros de Software na especificação de requisitos associados ao RNF Segurança, em aplicações Web desenvolvidas orientando-se pelo padrão arquitetural MVC..*"

- Intervenção: "*..utilização de notação específica, NFR Framework, para modelagem dos Grafos de Interdependências de Requisitos Não Funcionais centrados no critério de qualidade Segurança, especificando impactos, interdependências e operacionalizações cabíveis ..*"

- Saídas: "*.. catálogo de soluções centrado em Segurança e no padrão arquitetural MVC..*"

1.3 Justificativa

Observou-se que, mesmo com uma *baseline* arquitetural definida, os RNFs, chamados também de atributos ou critérios de qualidade, são negligenciados ou tratados de forma equivocada (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016). Entretanto, esses critérios evidenciam preocupações, as quais impactam na qualidade do software em desenvolvimento (SCHNEIDEWIND, 1990). Portanto, são necessários esforços para auxiliar os Engenheiros de Requisitos e os Engenheiros de Software na especificação desses RNFs, em particular em aplicações tipicamente comerciais, cujas arquiteturas são, normalmente, impostas pelas empresas de desenvolvimento.

1.4 Objetivos

1.4.1 Objetivo Geral

Definir um Catálogo de Segurança para o padrão arquitetural MVC, usando modelagem orientada a grafos de interdependências de RNFs, ou seja, levando em consideração as boas práticas acordadas na Engenharia de Requisitos Orientada à Meta (em inglês, *Goal Oriented Requirements Engineering* - GORE).

1.4.2 Objetivos Específicos

Com intuito de alcançar o objetivo geral, foram considerados relevantes os seguintes objetivos específicos:

- Investigar - na literatura: (i) formas recomendadas para lidar com o RNF Segurança em aplicações web desenvolvidas com base no padrão arquitetural MVC, permitindo identificar alternativas e operacionalizações para concretização desse RNF de forma satisfatória, e (ii) RNFs associados à Segurança, permitindo identificar as interdependências e os impactos entre eles;
- Elaborar o catálogo de SIGs, com base nos resultados da etapa de investigação (supracitada) bem como orientando-se pela notação do NFR *Framework*;
- Realizar a correspondência entre o catálogo e as camadas do padrão arquitetural MVC;
- Elaborar cenários e desenvolver aplicações web exemplo, no padrão MVC, orientando-se pelo catálogo. Essas aplicações web utilizadas como exemplos podem ser vistas como cenários de uso ou um estudo de caso, procurando apresentar, aos interessados, como o catálogo pode ser utilizado/instanciado.

1.5 Organização do Trabalho

Este trabalho está dividido nos respectivos capítulos, sendo eles:

- O Capítulo 2 **Referencial Teórico**, o qual explora os conceitos chave para elaboração desse trabalho, partindo da Engenharia de Requisitos e seus conceitos, perpassando por referenciais associados à Arquitetura de Software, e concentrando no NFR Segurança de Software. Ainda nesse capítulo, é acordado o conceito de *personas*, o qual foi utilizado na elaboração e na evolução do catálogo usando como base diferentes cenários de uso;
- O Capítulo 3 **Metodologia**, o qual foca nas questões da classificação da pesquisa e nos procedimentos metodológicos para o desenvolvimento desse trabalho;
- O Capítulo 4 **Suporte Tecnológico**, o qual apresenta as ferramentas de modelagem dos diagramas e mapas mentais, e as ferramentas de suporte à escrita utilizadas para a elaboração desse trabalho;
- O Capítulo 5 **Catálogo de Segurança**, o qual descreve o catálogo bem como detalha o grafo de interdependência entre as metas flexíveis (SIGs);
- O Capítulo 6 **Resultados Obtidos**, o qual apresenta a aplicação e a evolução do Catálogo de Segurança, considerando cenários de uso diferentes. Por fim, confere-se ênfase no mapeamento entre o Catálogo de Segurança e as camadas do Padrão Arquitetural MVC;
- O Capítulo 7 **Conclusão**, o qual acorda os resultados alcançados com o desenvolvimento desse trabalho bem como os principais trabalhos futuros almejados.

2 Referencial Teórico

Neste capítulo, serão apresentados os conceitos teóricos utilizados como base para o desenvolvimento do Catálogo de Segurança para o Padrão Arquitetural MVC.

Antes mesmo de apresentar os conceitos relevantes para o desenvolvimento do Catálogo de Segurança, é importante entender a definição de software. Este, segundo (SOMMERVILLE et al., 2003), pode ser entendido como programas de computador e toda documentação associada. Para realização do desenvolvimento de software, considerando as boas práticas da área, existem preocupações que merecem menção, como os RNFs e os aspectos arquiteturais inerentes à produção, sendo essas estabelecidas pela comunidade da Engenharia de Software (SOMMERVILLE et al., 2003). A Figura 1 apresenta o escopo de atuação desse trabalho, partindo da Engenharia de Software e apoiando-se, principalmente, em duas disciplinas:

- Engenharia de Requisitos, que estabelece um conjunto de tarefas e técnicas que auxiliam a promoção do entendimento dos requisitos, sendo uma disciplina detalhada na seção 2.1. Será observado que o Catálogo de Segurança concentra-se em uma abordagem orientada à meta. Portanto, tem-se a Engenharia de Requisitos Orientada à Meta. Essa abordagem trata os requisitos como metas a serem alcançadas, sendo esse estudo apresentado na subseção 2.1.1. Especificando ainda mais a área de atuação, como esse trabalho centraliza seus esforços nos RNFs, em particular para elaboração do catálogo, a subseção 2.1.2 define conceitos associados aos RNFs, preparando o leitor para a apresentação de um *framework* conceitual especificamente proposto para detalhamento de RNFs. Trata-se do NFR *Framework*, o qual será acordado na seção 2.2, e

- Desenho de Software, que se preocupa, dentre outras atividades, com o planejamento e a definição da Arquitetura de Software utilizada. Para fins desse trabalho, o foco será no Padrão Arquitetural MVC. Conceitos associados à Arquitetura de Software bem como ao Padrão Arquitetural MVC serão detalhados na seção 2.4 e na subseção 2.4.1, respectivamente.

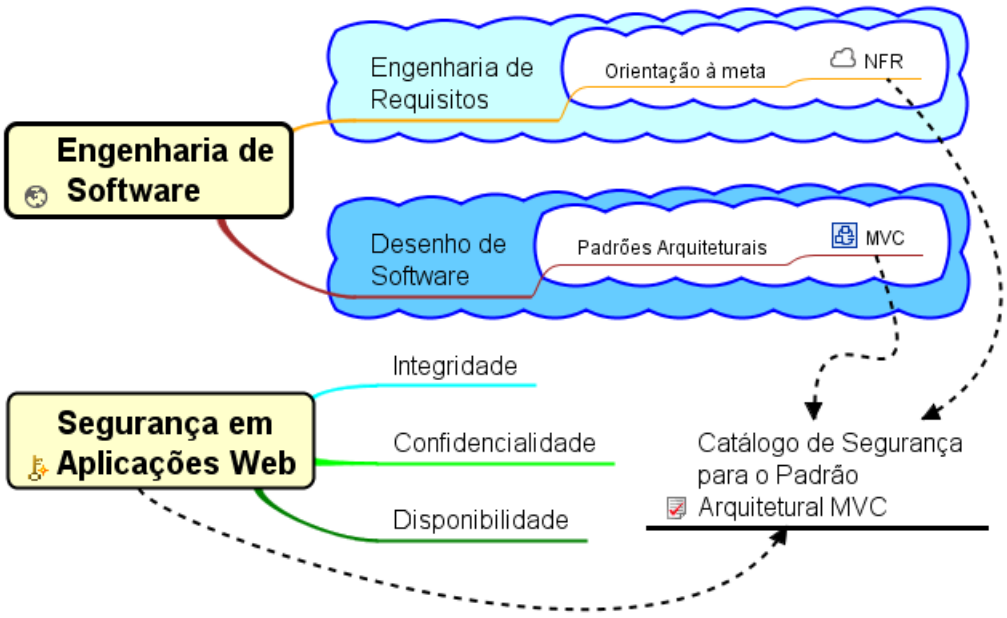


Figura 1 – Áreas de atuação do presente trabalho e conceitos associados.

Cabe ressaltar que o presente trabalho atua mais especificamente no requisito não funcional Segurança, dada a sua relevância, conforme pesquisa na literatura. Segundo (CHUNG et al., 2012), Segurança pode ser entendida como a satisfação de três conceitos: Integridade, Confidencialidade, e Disponibilidade. Portanto, o Catálogo de Segurança parte desses conceitos, seguindo a expressão 2.1. Segurança, sobre a ótica de um RNF, será tratada na seção 2.6. Por fim, o conceito de *Personas* será tratado na seção 2.7.

Integridade E Confidencialidade E Disponibilidade SATISFAZ Segurança (2.1)

2.1 Engenharia de Requisitos

Ao conjunto de tarefas e técnicas utilizadas para promover o entendimento dos requisitos é denominado Engenharia de Requisitos (PRESSMAN, 2011). No desenvolvimento de software, ela pode ser vista como uma disciplina importante da Engenharia de Software, que se inicia nas atividades de comunicação com os *stakeholders*, e continua até a entrega do produto, sendo adaptada de acordo com as necessidades do processo de desenvolvimento, do produto e dos *stakeholders* (PRESSMAN, 2011).

A Engenharia de Requisitos abrange sete fases sendo elas: Concepção, Levantamento, Elaboração, Negociação, Especificação, Verificação e Validação (PRESSMAN, 2011). De acordo com (KOTONYA; SOMMERVILLE, 1998), durante a execução das atividades nas fases da Engenharia de Requisitos, alguns problemas são relatados como: (i) os requisitos não refletem as reais necessidades do cliente, de acordo com o sistema a

ser desenvolvido; (ii) os requisitos são inconsistentes e/ou incompletos; (iii) a Engenharia de Requisitos é complexa e possui alto custo, principalmente, quando há necessidade de mudanças após os requisitos serem ditos acordados/elicitados entre as partes; e (iv) os requisitos são comumente interpretados de maneira errada pela equipe de desenvolvimento, diante do que foi solicitado pelo cliente. Apesar de parecer algo acordado em uma referência antiga da área. Tais problemas continuam ocorrendo em desenvolvimentos atuais, conforme apontado em palestras recentes de autores da área (MEIRA, 2015).

Boa parte dos modelos utilizados na Engenharia de Requisitos não possuem tratamento adequado para lidar com os critérios de qualidade. Logo, o tratamento desses critérios tem sido uma preocupação para a comunidade da Engenharia de Requisitos. Atualmente, existem esforços no sentido de aprimorar os modelos e as especificações desses requisitos, e são fortemente associados à comunidade de pesquisadores da Engenharia de Requisitos Orientada à Meta (GORE) (CHUNG et al., 2012). Neste trabalho, foi abordado o uso de uma notação específica para tratar RNFs, proposta por essa comunidade, no caso, trata-se do NFR *framework* (CHUNG et al., 2012).

O NFR *framework* é um *Framework* conceitual que procura lidar com os requisitos não funcionais, permitindo especificar os mesmos em um alto nível de abstração e, gradualmente, fornecendo insumos para que essa especificação seja trazida para um nível mais baixo de abstração. Nesse último nível, são especificadas as operacionalizações, as quais evidenciam alternativas para viabilizar a implementação desses requisitos no nível de código. Mais adiante na seção 2.2, serão apresentados detalhes dessa notação.

2.1.1 Engenharia de Requisitos Orientada à Meta

A Engenharia de Requisitos Orientada à Meta (GORE) tem-se popularizado nos últimos anos e foca seus esforços no tratamento dos requisitos, como metas a serem alcançadas (LAMSWEERDE, 2001). Seus modelos possuem como objetivo a modelagem da razão pela qual determinado requisito existe. Essa modelagem promove ao Engenheiro de Requisitos uma estratégia mais detalhada do problema, podendo encontrar uma solução mais adequada (LAMSWEERDE, 2001)(CHUNG et al., 2012). O GORE é uma abordagem cada vez mais reconhecida na comunidade de Engenharia de Requisitos (LAMSWEERDE, 2001).

Nas definições do GORE, existem as metas que são componentes essenciais. De acordo com (LAMSWEERDE, 2001), uma meta é definida como uma propriedade do sistema, a qual é expressa pelos *stakeholders* através de determinadas questões como: "**Porquê**" uma meta é exigida, "**como**" ela pode ser atingida, e "**quem**" será o responsável pela meta no sistema e no ambiente.

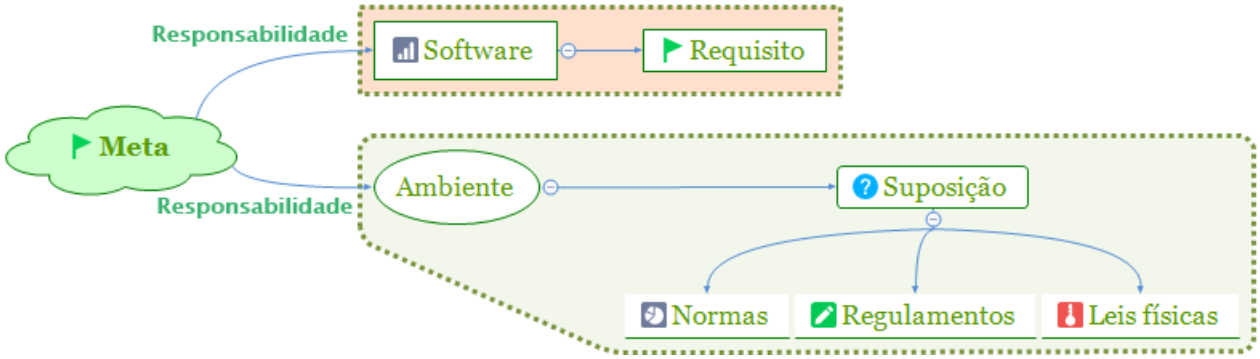


Figura 2 – Responsabilidades da meta sob o software e sob o ambiente.

Com base na Figura 2, pode-se entender como são os comportamentos das metas na abordagem GORE. Representada pelo símbolo de uma nuvem, uma meta, quando está sob a responsabilidade de um único software, pode se tornar um requisito. De forma semelhante, quando uma meta está sob a responsabilidade de um ambiente, ela pode se tornar uma suposição. Diferentemente de um requisito, uma suposição não pode ser aplicada pelo software. Mas, pode ser satisfeita devidos às normas, aos regulamentos organizacionais e às leis físicas(LAMSWEERDE, 2001). Esse trabalho focou apenas nas responsabilidades voltadas ao software.

2.1.2 Requisitos Não-Funcionais

A visão do software como um produto fez com que os aspectos que avaliam a qualidade de um software passassem a possuir maior atenção. Apenas satisfazer requisitos funcionais não é o suficiente. Logo, tem-se maior atenção para o RNFs, tais como: Segurança, Integridade, Disponibilidade, dentre outros (CYSNEIROS; LEITE, 1997).

Os RNFs podem ser entendidos como restrições sobre como os requisitos funcionais devem ser implementados, e determinam como o software deve realizar suas funções (SOMMERVILLE; SAWYER, 1997).

Júlio Leite, afirma em sua pesquisa que os RNFs impactam o produto software em qualidade e preço, pois segundo ele:

> "A não observância da necessidade de RNFs durante o processo de desenvolvimento do software, pode levar uma recodificação custosa e demorada o que consequentemente, eleva o preço do software." (CYSNEIROS; LEITE, 1997, p. 2)

Os RNFs devem ser analisados em toda sua magnitude para o desenvolvimento e a manutenção de um produto de software, sendo importante analisar os possíveis conflitos que podem ser gerados com outros RNFs bem como com os requisitos funcionais. Esses conflitos, uma vez não identificados em uma fase inicial do processo de desenvolvimento de

software, podem acarretar em problemas futuros, durante as atividades de implementação e implantação do software (CYSNEIROS; LEITE, 1997).

As classes de NFRs (ECKHARDT; VOGELSANG; FERNÁNDEZ, 2016) podem ser compreendidas como subtipos dos atributos de qualidade da Tabela 1, na qual são definidos de acordo com a (ISO/IEC-25010, 2011).

Tabela 1 – Descrição dos atributos de qualidade. Fonte: (ISO/IEC-25010, 2011).

Atributos de qualidade	**Descrição**
Funcionalidade	É a capacidade do software de promover funções que atendam às necessidades explícitas e implícitas.
Segurança	É a capacidade do software de apresentar níveis aceitáveis e riscos de danos a pessoas, negócios software, propriedades ou ambiente.
Confiabilidade	É a capacidade do software de manter o nível de desempenho especificado.
Usabilidade	É a capacidade do software de ser compreendido, de fácil aprendizagem, operável e atraente ao usuário.
Eficiência	É a capacidade software de apresentar desempenho apropriado, relativo aos recursos utilizados.
Portabilidade	É a capacidade software de ser transferido de um ambiente para o outro.
Manutenção	É a capacidade do software de modificado de forma eficiente e eficaz.

2.2 NFR Framework

O NFR *Framework* é um modelo intencional, criado para ajudar os especialistas a lidarem com requisitos não-funcionais, através de um grafo chamado de *Softgoal Interdependency Graphs* (SIGs). Neste tipo de grafo, os requisitos podem ser analisados, uma vez que o SIG permite uma visão vertical desde a estratégia de alto nível até os detalhes operacionais; possuindo operadores lógicos AND (E) e OR (OU) que promovem uma melhor tomada de decisão. Os RNFs são escritos através de uma notação formal. Essa notação permite comprovar a precisão e a completude de cada NFR (CHUNG et al., 2012).

O *framework* também pode ser orientado por processo, dado seu suporte às atividades e fases do processo de Engenharia de Requisitos. Nesse caso, sendo utilizado como complemento nas abordagens de desenvolvimento de software (CHUNG et al., 2012).

O NFR *Framework* utiliza o conceito de metas flexíveis. Por definição, uma meta flexível é uma condição ou um estado no mundo real que deseja ser alcançado, podendo assumir natureza subjetiva, uma vez que o RNF pode variar de acordo com o julgamento

de cada pessoa; e natureza relativa, uma vez que o RNF pode depender de algum tipo de relação com outro RNF (CHUNG et al., 2012).

A notação no SIG para a **meta flexível** é um símbolo similar ao de uma nuvem, conforme apresentado pela Figura 3. Esse símbolo possui pequenas variações na forma, podendo representar uma **operacionalização** (nuvem em negrito), e também uma **alegação** (nuvem tracejada).

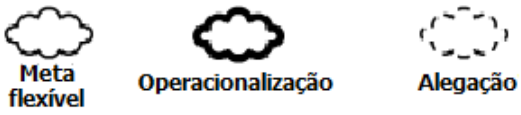


Figura 3 – Representações gráficas de Meta Flexível, Operacionalização e Alegação.

Figura 4 – Representações gráficas das *labels* em metas flexíveis.

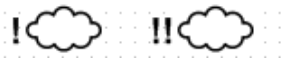


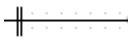
Figura 5 – Representações gráficas para o grau de prioridade da meta flexível.

Para facilitar o entendimento das decisões tomadas, a meta flexível possui *labels*, conforme apresentadas na Figura 3, que determinam o grau de satisfação para uma meta flexível, de acordo com um conjunto de decisões do projeto. As *labels* são: suficientemente satisfeita, fracamente satisfeita, negada, fracamente negada, indecidida e crítica (CHUNG et al., 2012), conforme apresentadas na Figura 4. A Figura 5 apresenta outra notação importante, e que representa o grau de prioridade para a meta flexível, é representada por "**!**" (média) ou "**!!**" (alta), permitindo especificar um grau de prioridade maior, sendo o ponto de afirmação desenhado à esquerda da nuvem. Tal notação aplica-se também a operacionalizações e alegações.

As relações que as metas flexíveis possuem umas com as outras são estabelecidas através de *links* de interdependências, sendo esses *links* que realizam o registro da decomposição das metas flexíveis em metas flexíveis mais específicas (filhos). De certa forma, a decomposição em outras metas contribui para o nível de satisfação da meta flexível mais genérica (pai).

Os *links* de interdependências também podem ser entendidos como tipos de contribuições, e possuem uma notação simbólica para cada tipo, e cada tipo possui ainda um significado. Tais tipos de contribuição são apresentados na Tabela 2, adaptados de (CHUNG et al., 2012).

Tabela 2 – Tipos de contribuições.

Símbolo	Descrição
	AND: Se todos os filhos são satisfeitos, o pai também será satisfeito.
	OR: Se qualquer filho é satisfeito, o pai também será satisfeito.
	MAKE: Pode ser tratado de forma semelhante ao AND, pois, se o filho for satisfeito, o pai pode ser satisfeito.
	BREAK: Fornece apoio negativo, pois, se o filho é satisfeito, o pai pode ser negado.
	HELP: Se todos os filhos são satisfeitos, o pai também será parcialmente satisfeito.
	HURT: Se todos os filhos são satisfeitos, o pai será fracamente negado.
	SOME+: Representa a existência de alguma contribuição positiva.
	SOME-: Representa a existência de alguma contribuição negativa.

Os *links* entre as metas flexíveis representam a contribuição que uma meta flexível tem com outra meta flexível, ou operacionalização, ou alegação. Logo, a Figura 6 apresenta a forma como as *labels* são propagadas, de acordo com seu tipo de relacionamento. A propagação das *labels* aplica-se a todos os tipos de representações da notação, sendo essas: metas flexíveis, operacionalizações ou alegações.

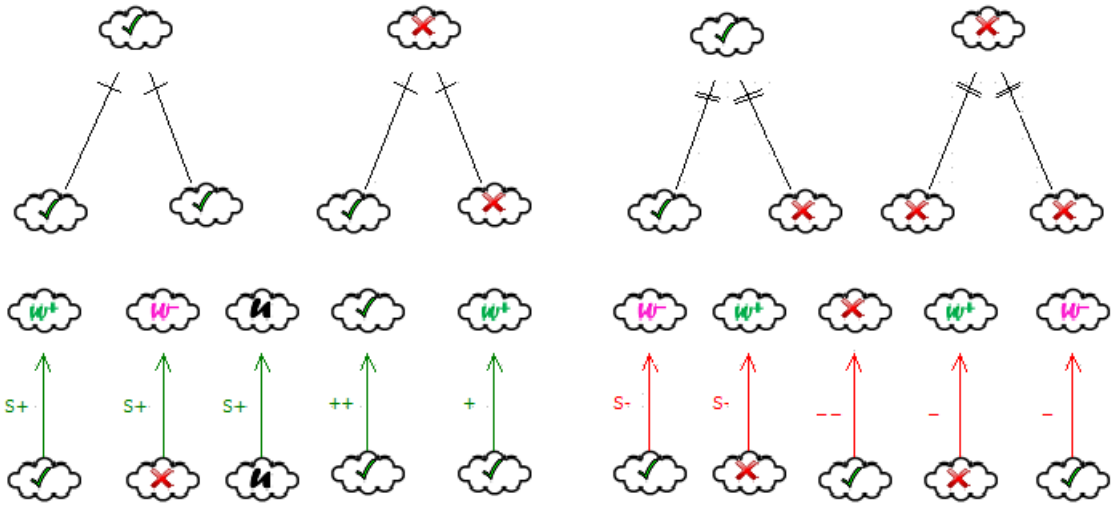


Figura 6 – Propagação das *labels* para os diferentes tipos de contribuição. Adaptado de (CHUNG et al., 2012).

A Figura 7 apresenta uma decomposição de uma meta flexível em diferentes níveis de abstração. Consideramos, nesse exemplo, o RNF: “manter as contas com boa segurança”. Usando a notação do NFR *Framework*, representou-se a meta flexível “segurança de contas”, no nível mais generalista do grafo. Em segundo nível, são especificadas as principais metas flexíveis que merecem ser consideradas para que a meta generalista seja "satisfeita", no caso: “Integridade de contas”, “Confidencialidade de contas” e “Disponibilidade de contas” (CHUNG et al., 2012).

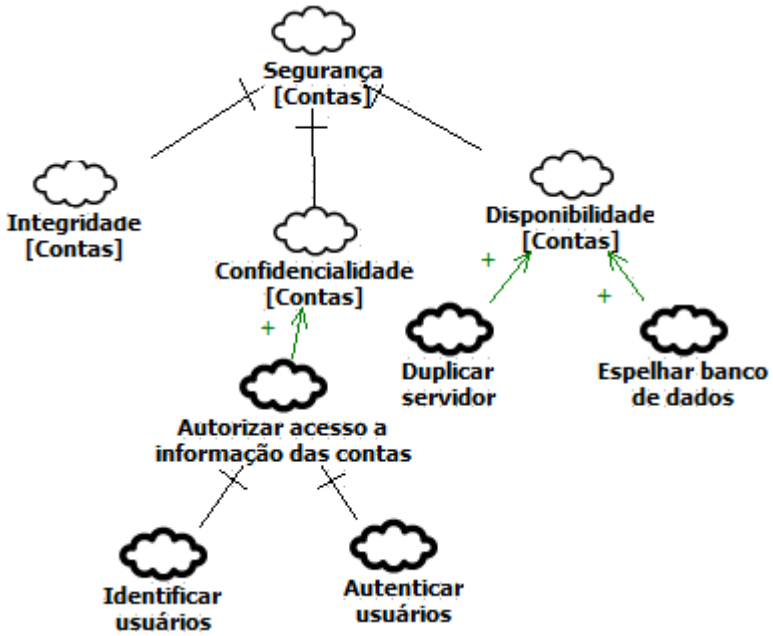


Figura 7 – Operacionalização de segurança de contas. Adapatado de (CHUNG et al., 2012), (AFFLECK; KRISHNA, 2012).

Confidencialidade de contas é operacionalizada em “Autorizar acesso à informação das contas“, que possui uma contribuição do tipo HELP com seu pai. Dada a relação do tipo AND, entre essa operacionalização e as operacionalizações “Identificar usuários“ e “Autenticar usuários“, tem-se que a operacionalização “Autorizar acesso à informação das contas“ será realizada, se ambas as operacionalizações, “Identificar usuários“ e “Autenticar usuários“, forem realizadas. Supondo que tudo tenha sido realizado com sucesso, tem-se que esse processo de operacionalização contribui positivamente - HELP (AJUDA) - a atingir “Confiabilidade de contas“. Como essa meta flexível está especificada em uma relação de AND com as metas flexíveis "Integridade de contas"e “Disponibilidade de contas“, pode-se dizer que em termos de “Confiabilidade de contas“, “Segurança de contas“ tende a ser “satisfeita“. Mas, resta ainda ponderar se "Integridade de contas"e “Disponibilidade de contas“ serão de fato “satisfeitas“. O modelo, da forma como está especificado, não permite avaliar tais aspectos, visto que representa apenas uma visão parcial dessas metas flexíveis.

Para Disponibilidade de contas, a mesma é operacionalizada em “Duplicar ser-

vidor" e "Espelhar banco de dados". Essas operacionalizações possuem contribuições do tipo HELP e mesmo pai. Para a meta flexível ser satisfeita, ambas as operacionalizações devem ser satisfeitas (AFFLECK; KRISHNA, 2012).

2.3 FURPS

O termo FURPS é um acrônimo que define um modelo de atributo de qualidade. Esse acrônimo é composto por cinco atributos de qualidade, sendo eles: (i) Funcionalidade (do inglês, *Functionality*), (ii) Usabilidade (do inglês, *Usability*), Confiabilidade (do inglês, *Reliability*), Desempenho (do inglês, *Performance*) e *Supportability* (manteve-se o termo em inglês, pois o mesmo significa considerar testabilidade, adaptabilidade, manutenibilidade, compatibilidade, configurabilidade, escalabilidade e outros aspectos associados, não sendo, portanto, possível encontrar um único termo em português capaz de representar todos esses aspectos). (UMAR; KHAN, 2011). Diante da definição exposta sobre o FURPS, as subseções abaixo apresentam o conceito de cada um dos cinco atributos de qualidade.

2.3.1 Funcionalidade

A Funcionalidade está diretamente ligada ao comportamento do software e à interação do usuário com ele (CINTRA, 2006). Representando os principais recursos do produto de software, as funcionalidades de um sistema nem sempre estão diretamente ligadas ao domínio (PRESSMAN, 2011).

Por definição, funcionalidade é a capacidade do software de promover funções que atendam às necessidades explícitas e implícitas, quando o software estiver sendo utilizado, de acordo com os cenários especificados (NBR-ISO/IEC-9126-1, 2003). De acordo com a ISO 25010, a funcionalidade pode ser entendida como: (i) grau em que o conjunto de funções do software abrange todas as tarefas especificadas e objetivos do usuário, (ii) grau em que as funções fornecem os resultados corretos com o grau de precisão necessário e (iii) grau em que as funções facilitam a realização de tarefas e objetivos específicos (GORDIEIEV et al., 2014).

2.3.2 Usabilidade

Quando se pensa em usabilidade, tem-se uma preocupação com a satisfação do usuário, levando em conta, por exemplo, sua experiência em manipular um computador, utilizar um software, questões de ergonomia, design da interface gráfica, e até mesmo o contexto para o qual a solução computacional está sendo desenvolvida (CINTRA, 2006).

Por definição, usabilidade é a capacidade do software de ser compreendido, aprendido, operado e atraente ao usuário, quando o software estiver sendo utilizado de acordo

com os cenários especificados (NBR-ISO/IEC-9126-1, 2003).

2.3.3 Confiabilidade

No caso da Confiabilidade, considera-se a prevenção de falhas, capacidade do software se recuperar de um erro, a precisão e o tempo entre falhas (*Mean Time Between Failure* (MTBF)) (CINTRA, 2006).

Por definição, confiabilidade é a capacidade do software em manter o nível de estabilidade esperado, garantindo, por exemplo, tolerância a falhas e até mesmo recuperação rápida diante da ocorrência dessas falhas, quando o software estiver sendo utilizado de acordo com os cenários especificados (NBR-ISO/IEC-9126-1, 2003).

2.3.4 Desempenho

Em desempenho, são relevantes vários aspectos. Alguns deles, como garantir tempo de recuperação adequado, são necessidades sombreadas e desejadas em outros critérios também, no caso, confiabilidade. Outros aspectos chave, inerentes ao escopo do critério desempenho, são: taxas de transferência e transmissão de dados (do inglês, *troughput*)[1], e consumo de recursos (energia, memória, processamento, cache, uso balanceado/otimizado do sistema operacional, entre outros (CINTRA, 2006).

Por definição, desempenho é o nível como o software atende ao conjunto específico de valores, especificado para cada característica atrelada à execução desse software (NBR-ISO/IEC-9126-1, 2003).

2.3.5 *Supportability*

Em *Supportability*, considera-se a capacidade de adaptabilidade, a possibilidade de realizar manutenções corretivas e evolutivas, a flexibilidade de configuração, de instalação e a internacionalização no sistema (CINTRA, 2006).

2.4 Arquitetura de Software

O conceito de *design* de software surgiu da década de 1970, quando os pesquisadores da época acreditavam que o *design* era uma atividade separada da implementação, exigindo notações, técnicas e ferramentas específicas. Foi somente a partir da década de 1990 que se utilizou o termo arquitetura de software, em contraste com *design* de software, para indicar noções de codificação, abstrações, padrões e treinamentos formais de arquitetos de software (PERRY; WOLF, 1992).

[1] Quantidade de dados que pode ser tranferidos de um lugar para o outro em um espaço de tempo previamente especificado.

O termo arquitetura de software é definido por (SHAW; GARLAN, 1996), como o entendimento do sistema em termos de seus componentes computacionais e os relacionamentos entre os componentes, os padrões que guiam a composição organizacional dos componentes, bem como as decisões de restrições arquiteturais.

As arquiteturas de software são projetadas de acordo com algum princípio de estruturação genérico, sendo esses princípios chamados de padrões arquiteturais (BUSCHMANN et al., 1996).

Os padrões arquiteturais são modelos para arquiteturas concretas de software, expressando a estrutura essencial para o desenvolvimento de um produto de software. O padrão fornece um conjunto de subsistemas predefinidos, incluindo regras e diretrizes para organizar as relações entre eles, além de especificar suas responsabilidades (BUSCHMANN et al., 1996).

A seleção de um padrão arquitetural é uma atividade fundamental ao desenvolver um sistema de software, sendo apropriado entender que cada padrão auxilia o desenvolvedor a alcançar uma propriedade de sistema específica. Dentre o conjunto dos padrões arquiteturais, existem padrões que auxiliam em propriedades similares, e podem ser agrupados em categoria, sendo elas (BUSCHMANN et al., 1996):

- da lama à estrutura;
- sistemas distribuídos;
- sistemas interativos, e
- sistemas adaptáveis.

A Tabela 3 apresenta detalhadamente os quatro tipos de categorias para os padrões arquiteturais, definidos por (BUSCHMANN et al., 1996), e seus respectivos padrões.

Tabela 3 – Descrição e padrões das quatro categorias de padrões arquiteturais.

Categoria	**Descrição**	**Padrões**
da lama à estrutura	Confere suporte ao desenvolvedor, evitando um "mar" de componentes ou objetos. Os padrões dessa categoria suportam uma decomposição controlada de uma tarefa geral em subtarefas cooperantes.	Padrão Layers Padrão Pipe-and-Filter Padrão Blackboard
sistemas distribuídos	O conjunto de sistemas interligados que compartilham processamento entre si.	Broker
sistemas interativos	O conjunto de padrões que suportam a estruturação de sistemas de software, os quais apresentam a interação humano-computador.	Model-View-Controller Presentation-Abstraction-Control
sistemas adaptáveis	O conjunto de padrões que suportam a adaptabilidade, a evolução tecnológica e a alteração nos requisitos funcionais.	Mircokernel Reflection

2.4.1 MVC: *Model-View-Controller*

O MVC divide a aplicação em três componentes: (i) a *Model*, a qual contém os dados e as funcionalidades, entendida como a camada de manipulação dos dados, (ii) a *View*, componente responsável por apresentar as informações para o usuário, e (iii) a *Controller*, responsável por receber e controlar as requisições de entrada do usuário, realizando o controle de qual *model* usar e qual *view* será mostrada ao usuário (BUSCHMANN et al., 1996).

2.4.1.1 *Model*

A *Model* realiza o encapsulamento dos dados da aplicação, e detalha todo o comportamento funcional da aplicação. Este componente também realiza a exportação dos procedimentos para que a *Controller* possa chamar esses procedimentos de acordo com as necessidades do usuário (BUSCHMANN et al., 1996).

É também na *Model* que ficam os registros de todos os outros componentes. Esses registros são mantidos através do mecanismo de troca de propagação (BUSCHMANN et al., 1996).

A Tabela 4 resume os principais aspectos da *Model*, tratando de suas responsabi-

lidades e seus colaboradores.

Tabela 4 – Responsabilidades e colaboradores da *Model* (BUSCHMANN et al., 1996).

Responsabilidades	Colaboradores
- Fornecer o núcleo funcional da aplicação - Registrar visualizações e controladores dependentes - Notificar componentes dependentes sobre mudanças de dados	*View* *Controller*

2.4.1.2 *View*

A *View* é o componente responsável por apresentar as informações ao usuário, recuperando os dados da *Model* através de procedimentos de atualização que são ativados pelo mecanismo de propagação de mudanças. A *View* também possui um relacionamento de 1 para 1, com a *Controller*, pois fornece procedimentos para a *Controller* manipular as exibições da *View* (BUSCHMANN et al., 1996).

A Tabela 5 resume os principais aspectos da *View*, tratando de suas responsabilidades e seus colaboradores.

Tabela 5 – Responsabilidades e colaboradores da *View* (BUSCHMANN et al., 1996).

Responsabilidades	**Colaboradores**
- Inicializar o controlador associado - Exibir informações ao usuário - Implementar o procedimento de atualização - Recuperar dados da *Model*	*Model* *Controller*

2.4.1.3 *Controller*

A *Controller* trata as entradas do usuário como eventos. As controladoras desses eventos recebem um pedido da *View*. Nesse caso, um procedimento específico na *Controller* trata esse pedido e realiza uma chamada de um procedimento na *Model*. A *Controller* basicamente traduz os pedidos de eventos para a *Model*, sendo que, a cada pedido existe uma *view* associada.

A Tabela 6 resume os principais aspectos da *Controller*, tratando de suas responsabilidades e seus colaboradores.

Tabela 6 – Responsabilidades e colaboradores da *Controller* (BUSCHMANN et al., 1996).

Responsabilidades	**Colaboradores**
- Aceitar entrada do usuário como eventos - Traduzir os eventos para a *Model* ou exibir os pedidos para a *View* - Se necessário, implementar prodecimentos de atualização	*Model* *View*

2.4.1.4 Fluxo de Interação entre os Componentes

Tão importante quanto a compreensão de cada componente é compreender o fluxo de interação entre esses componentes e suas interações diretas e indiretas. A Figura 8 representa os três componentes e suas interações, através da notação da UML. Conforme representado, dentro de um pacote da *Model* existe uma classe exemplo, cujo o nome é *Model.Class*, possuindo relações de contribuição com o pacote *Controller*, através da classe *ManipuladoraDeDados.Class*. De forma semelhante, existe a relação entre a *Controller* e a *View* entre as classes *ManipuladoraDeDados.Class* e *GUI.Class*. Os nomes atribuídos na diagramação são para auxiliar a compreensão.

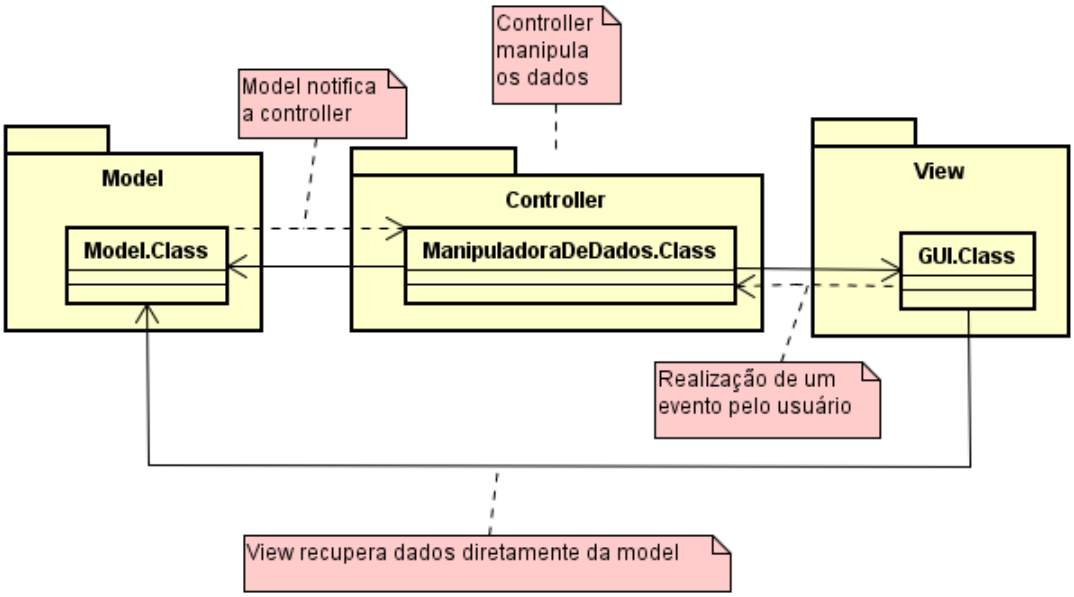


Figura 8 – Diagrama de classes representando a interação entre os componentes no padrão arquitetural MVC. Adaptado de (DURELLI; VIANA; PENTEADO, 2008).

A Figura 9 é um diagrama de sequência que representa o fluxo dos eventos em um cenário exemplo, no qual a *Controller* interage diretamente com a *Model* e a *View*, sendo que o usuário entra com alguma informação e aguarda o retorno da aplicação.

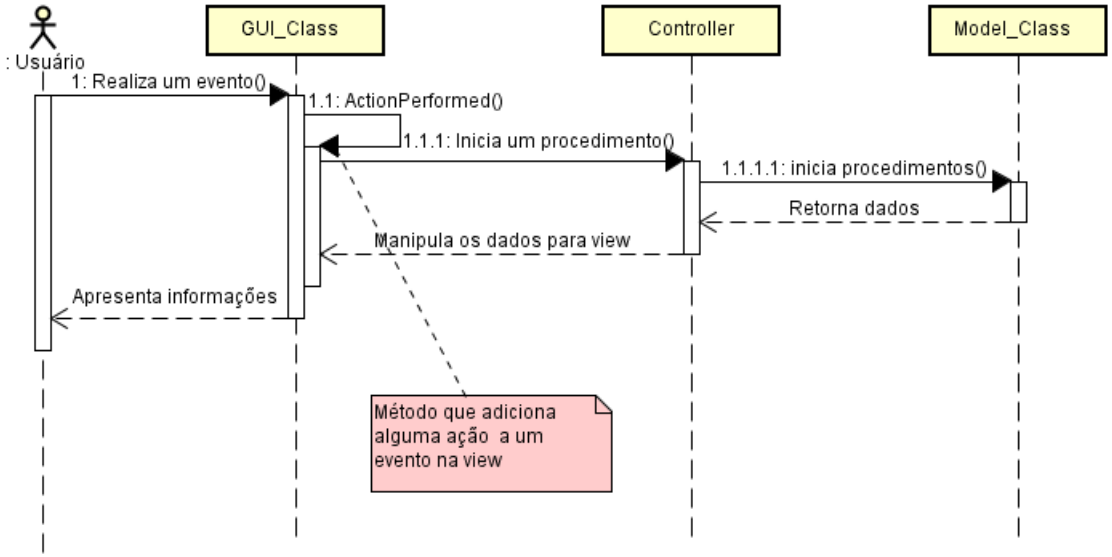


Figura 9 – Diagrama de sequência para o padrão arquitetural MVC. Adaptado de (DURELLI; VIANA; PENTEADO, 2008) e (BUSCHMANN et al., 1996).

2.5 ISO 27001

A ISO 27001 determina os padrões que podem ser aplicados em Sistemas de Gestão de Segurança da Informação (do inglês, *Information Security Management System*), intitulada como: " Tecnologia da informação - técnicas de segurança - sistemas de gerência da segurança da informação - requisitos". Tal norma, orienta como pode ser realizada a definição de um comitê multidisciplinar que possui como objetivo principal estabelecer as políticas de segurança dentro das organizações, determinando as medidas cabíveis dentro dos limites de atuação (ISO-27001, 2005).

A ISO 27001 define aspectos para segurança da informação, como: a preservação da confidencialidade, integridade e disponibilidade da informação, adicionando outras propriedades como autenticidade, responsabilidade, confiabilidade e auditória e controle (ISO-27001, 2005).

A relação da segurança da informação com a segurança de software se apoia nos mesmos pilares, sendo eles, disponibilidade, confidencialidade e integridade (CHUNG et al., 2012) (ISO-27001, 2005).

2.6 Segurança de software vista como um RNF

Tratando a Segurança como um RNF, pode-se entender a mesma como estando associada a vários outros RNFs. Dessa forma, podemos entender os RNFs associados à Segurança como sendo restrições, as quais realizam operacionalizações e satisfazem as metas de segurança, estabelecidas pelos Engenheiros de Requisitos e/ou Engenheiros de Software. Esse conjunto de RNFs devem expressar de maneira precisa e não ambígua as metas de segurança de uma aplicação, fornecendo uma especificação para alcançar a meta deseja (HALEY et al., 2006).

Existem variadas definições de Segurança. De forma simplificada, Segurança, no contexto de Segurança da Informação, significa proteger a informação (CHUNG et al., 2012). O presente trabalho fundamenta-se nos conceitos de (CHUNG et al., 2012) e (SULLIVAN; LIU, 2011) como base para definir Segurança, orientando-se por três conceitos conhecidos como CIA (*Confidentiality*, *Integrity*, *Availability*).

Diante do exposto, para a definição seguindo os fundamentos de (CHUNG et al., 2012), segurança se baseia em:

- **Confidencialidade** (do inglês, *Confidenciality*): Proteção da informação para evitar que as informações armazenadas ou transmitidas não sejam vistas ou interpretadas por terceiros, sendo somente o usuário principal e o destinatário. Deve-se prover essa proteção da informação através de algoritmos de criptografia (CHUNG et al., 2012) (SILVA; CUNHA, 2007).

- **Integridade** (do inglês, *Integrity*): Proteção contra qualquer tipo de atualização e/ou adulteração não autorizada. Certifica-se que essa proteção seja garantida contra todo tipo de modificação acidental ou maliciosa, garantindo que a informação percorra todo o trajeto entre o usuário principal e um destinatário (CHUNG et al., 2012) (SILVA; CUNHA, 2007).

- **Disponibilidade** (do inglês, *Availability*): Proteção contra a interrupção do serviço, no momento em que o usuário principal estiver necessitando da utilização do serviço (CHUNG et al., 2012) (SILVA; CUNHA, 2007).

O CIA pode ser abstraído em metas flexíveis associadas à Segurança. Logo após, essas metas flexíveis devem ser capturadas e organizadas em um catálogo de Segurança, promovendo um conjunto rico de alternativas e pontos de verificação para se proteger contra possíveis negligencias de pontos relevantes, e para assegurar informações em uma aplicação. Como apresentado na Figura 10, abaixo de cada conceito chave de Segurança, existem seus subtipos.

Apresentados em verde na Figura 10, estão outras características dos requisitos de Segurança, sendo um dos mais relevantes o "operacional", que possui como subtipo "segurança operacional" referindo-se, diretamente, à Segurança da Informação (CHUNG et al., 2012).

Os subtipos das "características dos RNFs", de acordo com (CHUNG et al., 2012), podem ser "ciclo de vida", "operacional[1]", "desenvolvimento[2]", "interna-externa[3]", "interna" e "externa".

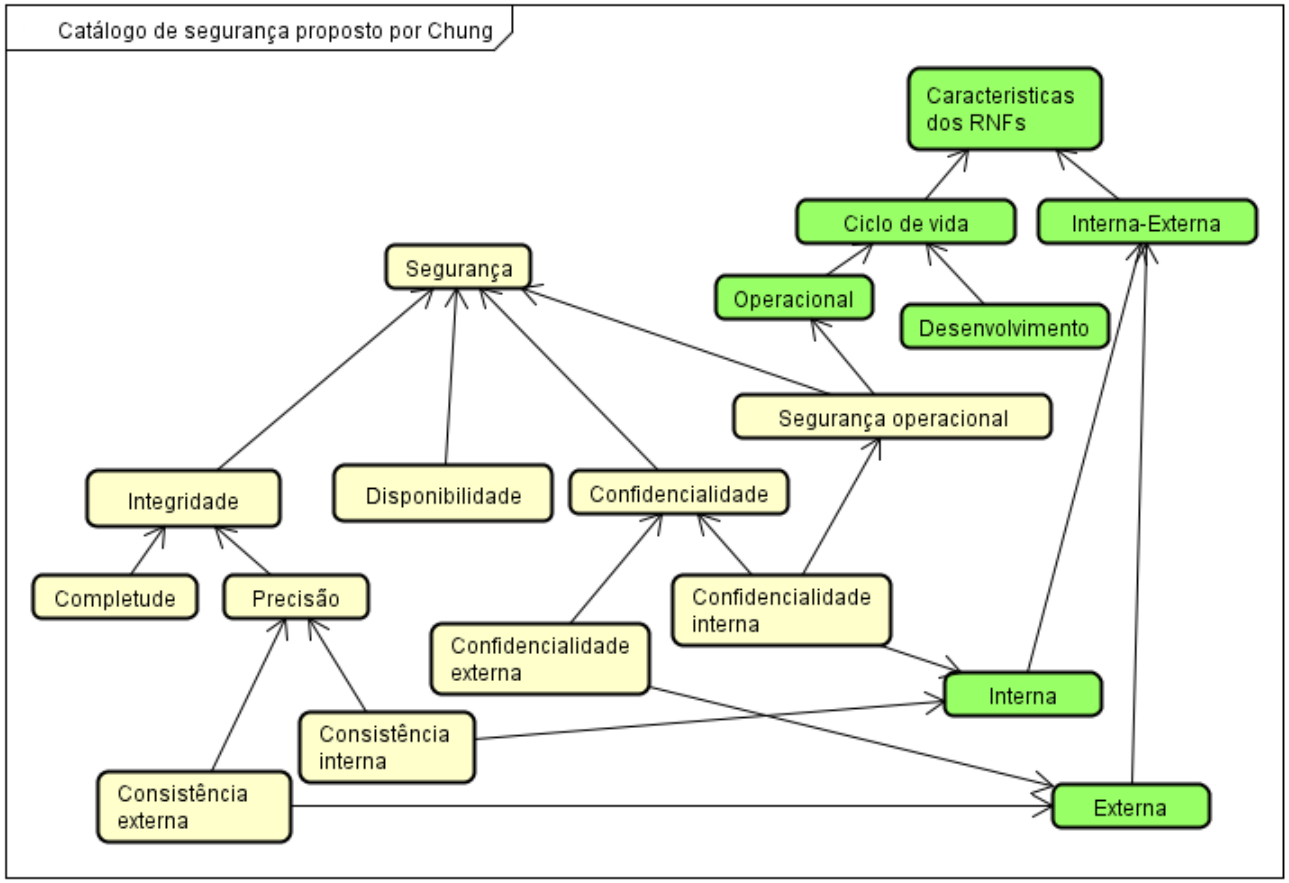


Figura 10 – Catálogo de Segurança. Fonte: (CHUNG et al., 2012).

É de extrema importância entender os subtipos de cada conceito. Portanto, as subseções a seguir fazem uma breve apresentação sobre os conceitos de Confidencialidade, Integridade e Disponibilidade.

2.6.1 Confidencialidade

De acordo com (REIS; MOTA; OLIVEIRA, 2001), em Confidencialidade, podem ser definidos conceitos sobre o tipo de Segurança, sendo eles:

[1] Operação realizada por uma aplicação de software em tempo de execução (CHUNG et al., 2012).
[2] Estágios de desenvolvimento (CHUNG et al., 2012).
[3] Refere-se a confidencialidade dos itens de informação do sistema (CHUNG et al., 2012).

- **Irrestrito**: tipo de informação pública;

- **Interna**: tipo de informação que seu acesso deve ser evitado por público externo. Caso este tipo de informação seja disponibilizado, por erro interno ou por ataque malicioso, não causa impacto algum ao mantenedor da informação;

- **Confidencial**: tipo de informação interna, a qual, uma vez disponibilizada ao público externo, por erro interno ou ataque malicioso, pode gerar vantagens a concorrentes e, consequentemente, perda de usuários/clientes;

- **Secreta**: tipo de informação interna, restrita a um conjunto específico de usuários, a qual, uma vez disponibilizada tanto ao público externo, quanto ao público interno não definidos, pode causar grandes danos. A integridade dessa informação deve ser mantida a qualquer custo.

A Confidencialidade possui os subtipos "confidencialidade interna" e "confidencialidade externa" conforme ilustra a Figura 10.

2.6.2 Integridade

Integridade pode ser definida através dos conceitos Completude e Precisão, por (CHUNG et al., 2012), onde:

Completude: garantir que o RNF esteja completo (CHUNG et al., 2012).

Precisão: pode ser entendida como qualquer atributo semântico que fundamenta uma informação (CHUNG et al., 2012), ou seja, a garantia de que um requisito da informação seja descrito com precisão de acordo com suas especificações;

- **Propriedade da precisão**: garantir que objetos sejam instanciados da maneira correta;

- **Valor de precisão**: garantir que os valores retornados pelas operações possuam a precisão desejada;

- **Precisão de um para um**: Garantir que um único objeto esteja ligado a uma única entidade do domínio, e

- **Consistência interna**: garantir que os valores do mundo real sejam correspondentes aos valores do sistema.

2.6.3 Disponibilidade

A disponibilidade pode ser entendida como o software se assegura diante de qualquer interrupção no serviço e que possa impedir o acesso ao mesmo, isto é, o quão disponível o sistema estaria para efetuar uma função solicitada (CHUNG et al., 2012).

2.7 *Personas*

Personas são especificações comportamentais que incorporam as metas e necessidades de usuários, esse conceito foi introduzido para auxiliar os desenvolvedores a entender melhor os usuários diante da necessidade de especificação de um sistema. *Personas* podem ser aplicadas na Engenharia de Software desde as atividades inerentes à Engenharia de Requisitos, até a entrega do produto propriamente dito (FORD et al., 2017).

Mais especificamente nesse trabalho, *Personas* foram utilizadas para a construção dos cenários, onde aplica-se os aspectos chave de segurança, privacidade e outras metas flexíveis de interesse, o que mitigou um pouco as dificuldades enfrentadas no entendimento bem como na especificação dos perfis de usuários compreendidos no escopo dessa pesquisa. Isso viabilizou, dentre outras contribuições, a modelagem de aspectos sociais e psicológicos, promovendo o entendimento dos perfis de usuários, e suas possíveis motivações, as quais permeiam suas ações (FORD et al., 2017) (RAMOS, 2013).

Ao longo do processo de construção de uma solução computacional - sob a perspectiva de um produto, tem-se a necessidade de imaginar como esse produto irá funcionar, e como seria possível aplicar esse produto (SOEGAARD; DAM, 2012). Nesse contexto, se os integrantes de uma equipe de desenvolvimento fizerem uso de *Personas*, o mesmos serão auxiliados, e terão maiores chances de manter o produto de acordo com os pontos de vista acordados, permitindo maior aderência desse produto às perspectivas dos usuários.Isso é possível, pois se pode imaginar como o produto será utilizado por uma Persona, a qual representa as metas e necessidades de um possível interessado nesse produto. Portanto, a intenção do uso de *Personas* não é descrever a persona em si, mas sim potencializar a capacidade da equipe no que tange a compreensão do produto, sendo esse idealizado sob diferentes pontos de vista (SOEGAARD; DAM, 2012).

No presente trabalho, apesar de se reconhecer a relevância do uso do conceito de *Personas* para mapeamento de aspectos sociais e psicológicos, não foi esse o maior foco. No caso, o uso de *Personas* contribuiu significativamente para construir o catálogo considerando pontos de vista diferentes, ora com maior olhar em Confiabilidade, ora em Integridade, ora em Disponibilidade, ora focado no Padrão Arquitetural MVC, dentre outras perspectivas. Esse exercício de considerar o catálogo sob diferentes pontos de vista possibilitou um aprofundamento de suas metas flexíveis, e mais ainda, um maior detalhamento das operacionalizações. Operacionalizações representam ações concretas, possíveis

de serem de fato alcançadas, implementadas e atendidas pelos especialistas. Uma vez atendidas, propagam-se suas contribuições em atendimento aos aspectos mais abstratos, no caso, as metas flexíveis. Diante dessas colocações, tem-se que o uso de *Personas*, em cenários de uso variados, contribuiu para a concretização do Catálogo de Segurança elaborado nesse trabalho.

Resumo do Capítulo

Neste Capítulo, foram abordados os conceitos fundamentais para a construção do Catálogo de Segurança, introduzindo conceitos chave da Engenharia de Requisitos, mais especificamente da Engenharia de Requisitos Orientada à Meta, com foco em metas flexíveis; bem como considerando conceitos chave de Arquitetura de Software, mais precisamente do Padrão Arquitetural MVC. Dada a relevância do critério de Segurança para o escopo do trabalho, no capítulo, tem-se uma seção dedicada à apresentação desse critério como um RNF. Conforme acordado, Confiabilidade, Integridade e Disponibilidade são metas flexíveis que impactam diretamente em Segurança. Por fim, comentou-se sobre o conceito de *Personas*, o qual representou um recurso relevante na elaboração bem como na evolução do Catálogo de Segurança sob diferentes pontos de vista, e considerando variados cenários de uso.

3 Metodologia

Neste Capítulo, é apresentada a classificação de pesquisa na seção 3.1, classificando-a de acordo com a sua abordagem, sua natureza, seus objetivos e procedimentos. A seção 3.2 apresenta a metodologia de desenvolvimento de software aplicada aos cenários. A seção 3.3 apresenta os procedimentos metodológicos utilizados na construção do trabalho.

3.1 Classificação da Pesquisa

A pesquisa científica pode ser classificada de acordo com sua abordagem, sua natureza, seus objetivos e seus procedimentos técnicos (GERHARDT; SILVEIRA, 2009). Portanto, nessa seção, procura-se enquadrar a classificação de pesquisa do presente trabalho, definindo-a quanto à Abordagem (seção 3.1.1); à Natureza de (seção 3.1.2), aos Objetivos (seção 3.1.3) e aos Procedimentos técnicos (seção 3.1.4).

3.1.1 Abordagem de Pesquisa

Segundo (GERHARDT; SILVEIRA, 2009), a pesquisa qualitativa preocupa-se com os aspectos da realidade que não podem ser quantificados, centrando-se na compreensão e explicação da dinâmica das relações sociais. Nela, há três características fundamentais para prosseguir com os estudos (MAZZOTTI, 1991):

- *Visão holística:* a compreensão de um comportamento ou evento só é possível com o entendimento das inter-relações que surgem dentro do contexto da pesquisa;
- *Abordagem indutiva:* o pesquisador parte de observações mais livres, deixando que as dimensões e categorias de interesse surjam progressivamente durante todo o processo de coleta e análise dos dados;
- *Investigação naturalística:* a intervenção do pesquisador, no contexto observado, é reduzida ao mínimo.

A abordagem de pesquisa quantitativa possui perspectiva no pensamento lógico positivista, tentando enfatizar o raciocínio dedutivo, as regras da lógica e os atributos mensuráveis registrados por um indivíduo (GERHARDT; SILVEIRA, 2009).

A abordagem de pesquisa qualitativa foi utilizada na execução deste trabalho junto com aspectos da pesquisa quantitativa, formando então, uma abordagem híbrida, orientado-se mais pela visão qualitativa. Considerando que, para a execução da pesquisa,

o autor tem a necessidade de realizar a descrição dos principais subtipos de RNFs de Segurança, visando a estruturação de um Catálogo de Segurança utilizando o NFR *Framework*, bem como, compreender e explicar os impactos entre os RNFs no Padrão Arquitetural MVC.

3.1.2 Natureza de Pesquisa

Essa pesquisa foi caracterizada como uma **pesquisa aplicada**, pois os conhecimentos gerados são voltados para a aplicação em cenários de uso que simulam situações reais ou mesmo que são situações reais (GERHARDT; SILVEIRA, 2009), visando auxiliar os especialistas na especificação dos RNFs de segurança, quando uma arquitetura é imposta.

3.1.3 Objetivos de Pesquisa

Quanto aos objetivos, essa pesquisa pode ser classificada como **pesquisa explicativa**, pois, segundo (GIL, 2002), "... este tipo de pesquisa preocupa-se em identificar os fatores que determinam ou que contribuem para a ocorrência dos fenômenos...", ou seja, explicou-se como o Padrão Arquitetural MVC pode ser impactado pelos RNFs, demonstrado pelo Catálogo de Segurança.

3.1.4 Procedimentos Técnicos de Pesquisa

A **pesquisa bibliográfica** foi um dos procedimentos técnicos de pesquisa utilizados, como é evidente no Capítulo 2, onde foi apresentado todo o levantamento teórico que serviu de insumo para a execução desse trabalho.

Fonseca descreve o procedimento técnico de pesquisa como:

> "*... esse tipo de procedimento trata-se do levantamento de referências teóricas já analisadas, e publicadas por meio de escritos e eletrônicos, como livros, artigos científicos, páginas de web sites. Qualquer trabalho científico inicia-se com uma pesquisa bibliográfica, que permite ao pesquisador conhecer o que já estudou sobre o assunto*". (FONSECA, 2002, p.35)

Outro procedimento técnico de pesquisa adotado foi a **pesquisa-ação**, pois trata-se da participação planejada do autor em uma situação problemática a ser investigada (FONSECA, 2002). O autor participou ativamente do levantamento dos conceitos teóricos, adquirindo consigo uma série de conhecimentos que serviu como base para construção do Catálogo de Segurança e que foi validado através da criação e da aplicação em cenários de uso.

3.2 Metodologia de desenvolvimento de software aplicada aos cenários

A metodologia de desenvolvimento de software utilizada foi o ***scrum* com adaptações**. Trata-se de uma metodologia de desenvolvimento ágil utilizada para a gestão e o planejamento de projetos de software (SCHWABER; BEEDLE, 2002).

O *scrum* trata os projetos de forma iterativa e incremental, entregando ao final de cada *sprint*[1] uma versão funcional do software com o incremento de uma nova funcionalidade (SCHWABER; BEEDLE, 2002). Para a execução do desenvolvimento do software nos cenários, cada cenário possuiu a duração de três semanas, e foi tratado como uma *sprint*, iniciando em Agosto e encerrando em Novembro.

Outro conceito do *scrum* utilizado foi o *product backlog*[2]. Para execução deste trabalho, foi aplicado o conceito de *product backlog* para os cenários.

Com base no *product backlog*, o *scrum team*[3] realizou o planejamento da *sprint*, de acordo com as funcionalidades mais relevantes, tendo como base o contexto definido em cada cenário.

As reuniões de retrospectiva e revisão de *sprint* ocorreram duas vezes dentro da *sprint*, foram reuniões semanais de ponto de controle com os orientadores as quais apresentava-se o que foi feito, o que não foi feito, e o porquê não foi feito, no decorrer da *sprint*. Algumas reuniões foram realizadas de forma presencial, com os orientadores. A intenção dessas reuniões era apresentar o que estava sendo feito.

O Kanban é um método ágil que promove ao *scrum team* e aos *stakeholders* a visão geral da gestão do estado de implementação das funcionalidades (PRIKLADNICKI; WILLI; MILANI, 2014). Foi utilizado para controle visual do fluxo das atividades inerentes ao desenvolvimento da solução, dividindo suas colunas em: “A fazer”, “Fazendo” e “Feito”.

[1] Curto período de tempo em que são realizadas as atividades de desenvolvimento de software, é indicado que seja de duas a quatro semanas (SCHWABER; BEEDLE, 2002).

[2] Lista de funcionalidades a serem implementadas no projeto de software, não necessita estar preenchida por completo no inicio do projeto, pois essa lista cresce a medida que se aprende mais sobre o produto de software (SCHWABER; BEEDLE, 2002).

[3] Um time scrum é definido por uma equipe de pessoas variando de seis a dez pessoas (SCHWABER; BEEDLE, 2002). No presente trabalho o *scrum team* sofre adaptação para somente uma única pessoa “o autor”, mas o termo será utilizado para referir-se diretamente ao autor.

3.3 Procedimentos Metodológicos

Para o desenvolvimento deste trabalho, foi realizado um conjunto de atividades para que o Catálogo de Segurança pudesse ser elaborado e aplicado em cenários. Portanto, essas atividades foram modeladas em notação BPMN, com intuito de detalhar o fluxo de execução.

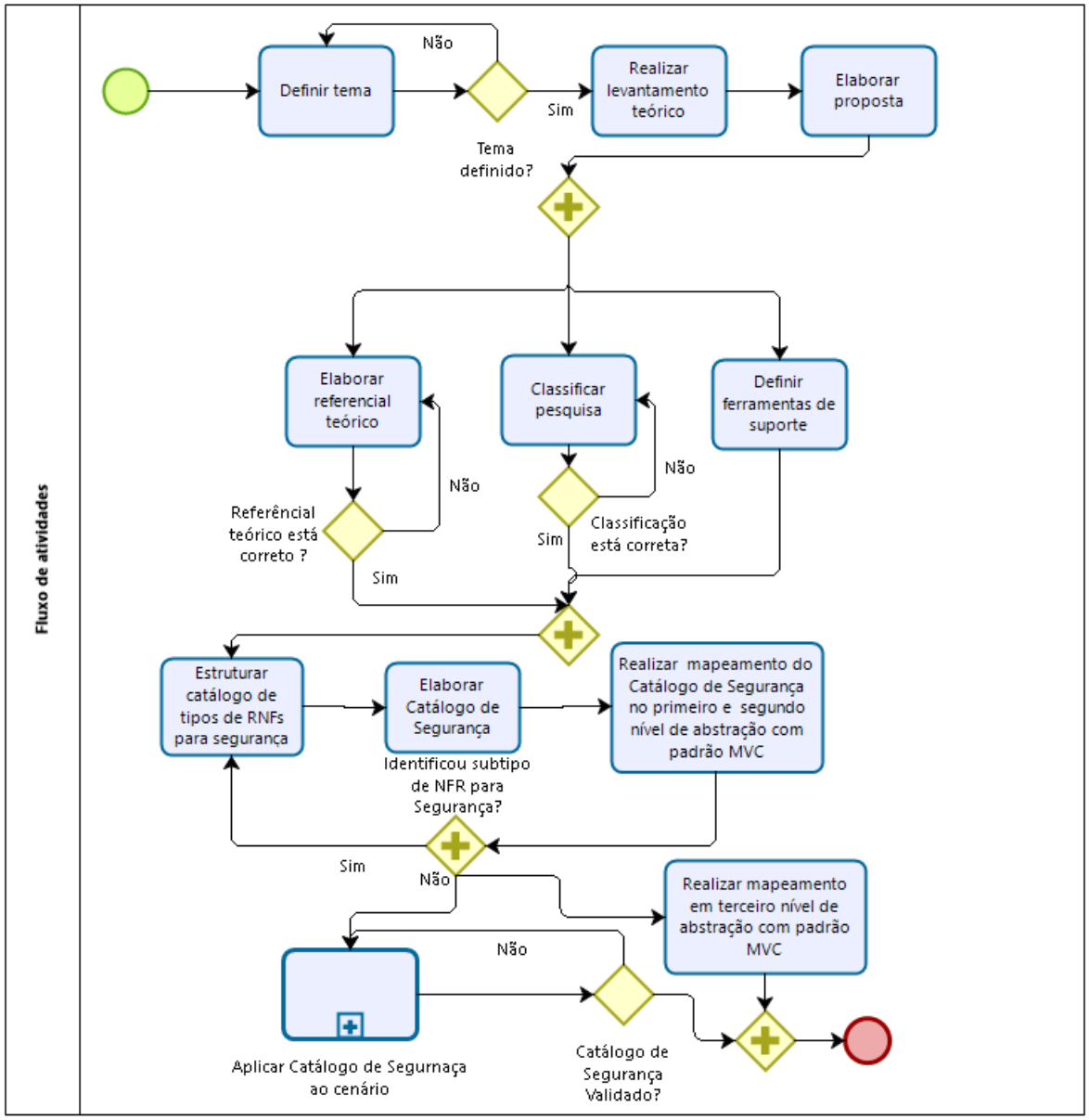


Figura 11 – Atividades realizadas.

Segue abaixo a descrição das atividades:

- **Definir tema**: em conjunto com os professores orientadores dessa monografia, essa atividade teve como principal objetivo o refinamento sobre os conceitos teóricos referentes à área de interesse do autor;

- **Realizar levantamento teórico**: iniciou-se com base nos resultados obtidos na execução da atividade anterior, seguindo as orientações dos orientadores. Discutiu-se sobre os aspectos fundamentais do GORE, os *frameworks* que se apoiam nos conceitos do GORE, os conceitos de Arquitetura de Software e os atributos de qualidade mais relevantes para o mercado e para a academia;

- **Elaborar proposta**: com o conhecimento das necessidades da área de atuação desse trabalho, iniciou-se a elaboração da proposta de pequisa, acordando que seria utilizado o NFR *Framework*, bem como, que seria aplicado ao mercado, tomando Segurança como o principal atributo de qualidade;

- **Elaborar referencial teórico**: foi realizada a redação dos conceitos teóricos a serem aplicados no desenvolvimento deste trabalho;

- **Classificar pesquisa**: redação da classificação da pesquisa em suas dimensões, sendo especificada quanto (i) à abordagem, (ii) à natureza, (iii) aos objetivos, e (iv) aos procedimentos técnicos;

- **Definir ferramentas de suporte**: definição e descrição das ferramentas utilizadas no desenvolvimento deste trabalho;

- **Estruturar catálogo de tipos de RNFs para Segurança**: com base nos conceitos de Segurança apresentados por Lawrence Chung, Brian A. Nixon, Eric Yu e John Mylopoulos, no livro: *NON-FUNCTIONAL REQUIREMENTS IN SOFTWARE ENGINEERING* (CHUNG et al., 2012), iniciou-se a elaboração do catálogo de tipos de RNFs para Segurança, expandindo o catálogo com base em definições de RNFs correlatos;

- **Elaborar Catálogo de Segurança**: a partir do Catálogo de Segurança apresentado na Figura 10, iniciou-se o processo de identificação das metas flexíveis que podem ser suficientemente satisfeitas, orientando-se também pelo livro supracitado;

- **Realizar mapeamento do Catálogo de Segurança no primeiro e segundo nível de abstração com o padrão MVC**: iniciou-se a comparação das metas flexíveis que possuíam relação com o Padrão Arquitetural MVC, a fim de verificar se estas geram impactos, positivos ou negativos, na Segurança dos dados e da arquitetura. Durante a execução dessa atividade, caso ocorresse a identificação de outro subtipo de RNF para Segurança, a atividade "Estruturar catálogo de tipos para Segurança" seria realizada novamente e, consequentemente, as atividades seguintes, de acordo com a Figura 11;

- **Aplicar Catálogo de Segurança ao cenário**: esta macroatividade respeitou a metodologia de desenvolvimento de software descrito na seção 3.2. Haja vista que,

em cada ciclo, foi definido um cenário com a descrição da persona, identificando, caso necessário, uma possível solução e realizando a aplicação do catálogo. Com isso, realizou-se o desenvolvimento do software, evidenciando a aplicação do catálogo e possibilitando a análise do quanto a Segurança do software pode ser satisfeita, e

- **Realizar mapeamento em terceiro nível de abstração com padrão MVC**: esta atividade foi executada em paralelo com o processo de desenvolvimento das aplicações no cenário, no qual foi possível aplicar o Catálogo de Segurança e realizar o detalhamento em terceiro nível de abstração. Esse detalhamento foi focado no mapeamento das operacionalizações com as camadas do padrão arquitetural MVC.

3.3.1 Cronograma das Atividades

O cronograma apresentado na Tabela 3.3.1, organizado em meses, expõe como se deu a execução das atividades deste trabalho.

Tabela 7 – Cronograma de execução do trabalho.

Ano	**2017**					**2018**							**2019**		
Atividade/Mês	**Ago**	**Set**	**Out**	**Nov**	**Dez**	**Jan**	**Fev**	-	**Ago**	**Set**	**Out**	**Dez**	**Jan**	**Fev**	**Mar**
Definir tema	x							-							
Elaborar proposta	x	x						-							
Elaborar referencial teórico	x	x	x	x	x			-							
Classificar pesquisa			x	x	x			-							
Definir ferramentas de suporte			x	x	x			-							
Estruturar Catálogo de Segurança					x	x		-							
Elaborar Catálogo de Segurança					x	x		-							
Realizar Mapeamento do Catálogo de Segurança no primeiro e segundo nível de abstração com padrão MVC.						x	x	-							
Aplicar Catálogo de Segurança ao cenário								-	x	x	x	x	x	x	
Realizar mapeamento em terceiro nível de abstração								-	x	x	x	x	x	x	x

Observa-se que foram utilizados, além do prazo base para a construção do Catálogo de Segurança, os meses de Janeiro/2018 e Fevereiro/2018. Isso ocorreu para maior aprofundamento do tema, o qual demandou a leitura de vários materiais bibliográficos. Adicionalmente, as notações envolvidas nessa pesquisa são complexas, o que exige de um cuidado maior na escrita com o objetivo de promover uma melhor compreensão por parte dos interessados.

Com o Catálogo de Segurança concluído, iniciou-se o ciclo de aplicação do mesmo em cada cenário. Essa aplicação ocorreu entre o período de Agosto de 2018 a Fevereiro de 2019, haja vista que alguns cenários eram contextos reais. Dessa forma, somente em Março de 2019, foi possível obter o mapeamento no terceiro nível de abstração.

Resumo do Capítulo

Neste capítulo, foi detalhada na, seção 3.1, a classificação de pesquisa de acordo com seus tipos, sendo estes: (i) abordagem, se tratando de uma pesquisa híbrida, orientando-se mais pela visão qualitativa, a (ii) Natureza, caraterizada como uma pesquisa aplicada, o (ii) objetivo, caracterizado como uma pesquisa explicativa, e os (iv) procedimentos técnicos, caracterizados como uma pesquisa bibliográfica e a pesquisa-ação. Mais adiante, na seção 3.2, foi descrita a adaptação do *Scrum* como metodologia de desenvolvimento de software e o Kanban para acompanhamento visual do andamento do desenvolvimento das aplicações nos cenários. Por fim, na seção 3.3, o fluxo de atividades e o cronograma realizados na execução deste trabalho foram detalhados.

4 Suporte Tecnológico

Neste Capítulo, são apresentadas as ferramentas de software que apoiaram o desenvolvimento desse trabalho. A seção 4.1 apresenta uma breve descrição das ferramentas utilizadas na modelagem dos diagramas e mapas mentais. A seção 4.2 apresenta as ferramentas de apoio à escrita, e para a realização do controle de versão. A seção 4.3 apresenta as ferramentas utilizadas no desenvolvimento de diferentes cenários de uso para aplicação e evolução do Catálogo de Segurança.

4.1 Ferramentas de Modelagem

Parte das contribuições desse trabalho é dedicada à elaboração do Catálogo de Segurança, o qual é apoiado no referencial teórico de GORE, conforme acordado na seção 2.1.1. Visando a representação desse catálogo em diferentes níveis de abstração, foram utilizadas algumas notações, no caso, UML, NFR *Framework*, Mapa Mental (XMIND.LTD, 2017) e BPMN (WHITE, 2004).

A UML foi utilizada para especificação de alguns detalhes atrelados, principalmente, ao uso do Padrão Arquitetural MVC, evidenciando as correlações entre suas camadas, conforme consta na seção 2.4.1.4. O NFR *Framework* foi utilizado como base para especificação do Catálogo de Segurança como um todo, fazendo-se uso das abstrações de metas flexíveis, operacionalizações, alegações, dentre outras mais específicas dessa notação. Mapas mentais permitiram documentar *brainstorming*, o qual representou uma técnica de elicitação muito utilizada ao longo desse trabalho. Por fim, para ilustrar as típicas atividades que conduziram a realização desse trabalho, optou-se por representar os fluxos de atividades em BPMN. Diante do exposto, foram necessários suportes tecnológicos que apoiassem as modelagens supracitadas, nas diferentes notações, conforme acordado a seguir:

- **Astah Professional**: é uma ferramenta de modelagem de diagramas dinâmicos e estáticos, que se orienta pelas notações UML 2.x, *Entity Relationship Diagram* (ERD), *Data Flow Diagram* (DFD), fluxogramas, mapas mentais, e apoia a Engenharia Reversa para as linguagens Java, C# e C++ (ASTAH, 2017). A versão utilizada foi a v7.2.0, com licença de estudante, para modelagem dos diagramas da UML e mapas mentais;

- **StarUML**: é um software *open source* para modelagem de diagramas na notação UML/*Model Driven Architecture* (MDA). A proposta da ferramenta é tornar-se uma ferramenta gratuita que substitua as plataformas comerciais (STARUML, 2017). A

versão utilizada foi a versão v1.0, pois é compatível com o*plug-in* RE-*Tools*, o qual permite a modelagem do SIGs para o NFR Framework;

- **Xmind**: é um software proprietário para criação de mapas mentais e documentação de *brainstorming* (XMIND.LTD, 2017). A versão utilizada foi o XMind 8 Update 2, permitindo a criação de mapas mentais;

- **Bizagi Modeler**: é uma ferramenta gratuita utilizada para modelagem de processos de negócio, utilizando a notação BPMN. A versão utilizada foi a versão 3.1, para a modelagem do fluxo de atividades que conduziram o desenvolvimento desse trabalho (BIZAGI, 2018), e

- **RE-*Tools***: é um conjunto de ferramentas *open source* utilizado para a modelagem de diferentes aspectos das organizações e de seus sistemas, durante a realização das atividades da Engenharia de Requisitos, conferindo apoio às notações do (i) NFR *Framework*, (ii) i* (HORKOFF; YU, 2006), (iii) *Knowledge Acquisition in autOmated Specification* (KAOS) (LAMSWEERDE, 2001), (iv) *Problem Frames* (JACKSON, 2005), (v) UML e (vi) BPMN (RE-TOOLS, 2017) (SUPAKKUL; CHUNG, 2012).

  Esse conjunto de ferramentas vem sendo utilizado para o ensino da Engenharia de Requisitos em cursos na Universidade de Trento (Itália), Universidade do Texas em Dallas (Estados Unidos) e na Universidade de Brasília (Brasil). Além disso, essas ferramentas constam em pesquisas realizadas em: Brasil, Austrália, Canadá, China, França, Reino Unido e Estados Unidos (SUPAKKUL; CHUNG, 2012).

  No presente trabalho, a versão do *plug-in* utilizada foi a v3.0.2, para modelagem do gráfico de SIGs.

4.2 Ferramentas para Desenvolvimento do Trabalho

A escrita da monografia também demandou suportes tecnológicos específicos, sendo os principais:

- **Git**: é um sistema *open source* que permite manter o controle de versão distribuído (GIT, 2017). A versão utilizada foi a v2.13.0, para realizar, o controle de versão da parte escrita do trabalho;

- **GitHub**: o GitHub é uma plataforma de hospedagem de código para controle de versão utilizando o Git, através do GitHub, é possível criar uma conta para armazenar o código fonte de projetos em repositórios gratuitos e privados (GITHUB, 2017). Utilizado para hospedar o repositório público para o controle de versão da monografia, sendo a versão utilizada a v1.0.13, e

- LaTeX : é um sistema que permite a elaboração de textos em alta qualidade, através do programa de diagramação de textos TEX (LATEX, 2017). A versão utilizada foi a v5.6.2, para documentação da parte escrita.

4.3 Ferramentas para o Desenvolvimento da Aplicação Web

Por fim, constam os suportes tecnológicos que viabilizaram o desenvolvimento dos cenários de uso, nos quais o Catálogo de Segurança foi aplicado. Isso permitiu evoluir o catálogo, obtendo versões do mesmo cada vez mais refinadas. Cabe ressaltar que, o Git e o GitHub também foram utilizados para controle das versões de código, oriundas dos produtos de software de cada cenário de uso. Seguem esses suportes:

- **Linux Mint 18.3 Sylvia**: é uma distribuição linux, baseada no Ubuntu 16.04 LTS, sendo a terceira e última versão do ciclo da série 18 (LINUX-MINT-COMMUNITY, 2018). Essa versão do sistema operacional foi utilizada em atendimento ao ambiente de desenvolvimento dos cenário, A versão utilizada encontra-se disponível em <https://linuxmint.com/download.php>;

- **Sublime Text:** é um editor de textos multiplataforma. Esse editor possui recursos que facilitam a escrita de código, como: Minimap, edição de textos em multi-painel, salvamento automático, autocompletar, correspondência entre parenteses, dentre outros (SUBLIME-TEXT, 2018). A versão utilizada foi a versão 3.0;

- **Ruby:** é uma linguagem de programação interpretada e multiparadigma, desenvolvida por Yukihiro "Matz" Matsumoto, unificando caraterísticas de suas linguagens favoritas (Perl, Smalltalk, Eiffel, Ada e Lisp), para formar uma linguagem que equilibra a programação funcional com a programação imperativa (RUBYCOMMUNITY, 2018). A versão da linguagem utilizada foi a versão 2.5.0;

- **Ruby on Rails - (RoR):** é um *framework open source* de desenvolvimento de aplicações web, escrito em **Ruby**, e baseado no Padrão Arquitetural MVC (RAILS-COMMUNITY, 2018). A versão do *framework* utilizada foi versão 5.1.5;

- **MySQL**: de acordo com a Oracle, o MySQL é o Sistema de Gerenciamento de Banco de Dados (SGBD) *open source* mais conhecido do mundo, além de ser uma das principais tecnologias de SGBD para desenvolvimento de aplicações web (ORACLE, 2018). A versão do MySQL utilizada foi o MySQL *Community Server*, sob licença GPL, disponível na versão 5.7.21, e

- **Bootstrap:** é um *framework open source* de desenvolvimento dos componentes responsivos para interface gráfica de aplicações web, que utiliza as Linguagens HTML, CSS e JavaScript (BOOTSTRAP, 2018). A versão utilizada foi a versão 4.0.0.

Resumo do Capítulo

Neste capítulo, foram descritas as principais ferramentas que apoiaram o desenvolvimento do trabalho. Na seção 4.1, foram apresentadas as versões e a utilização de cada ferramenta; a seção 4.2 descreveu as ferramentas utilizadas no desenvolvimento da parte escrita e o controle de versão da mesma, e a seção 4.3 acordou as ferramentas que foram utilizadas no desenvolvimento dos cenários de aplicação do catálogo.

A tabela 8 resume as principais ferramentas que foram utilizadas no trabalho.

Tabela 8 – Resumo das ferramentas de apoio.

Ferramenta	**Versão**	**Apoio**	**Aplicabilidade**
Astah Professional	v7.2.0	Modelagem	Modelagem de diagramas dinâmicos e estáticos.
StarUML	v1.0	Modelagem	Modelagem de diagramas na notação UML/MDA.
Xmind	XMind 8 Update 2	Modelagem	Modelagem de mapas mentais.
Bizagi Modeler	v3.1	Modelagem	Modelagem de processos de negócio, utilizando a notação BPMN.
RE-Tools	v3.0.2	Modelagem	Modelagem do gráfico de SIGs.
Git	v2.13.0	Controle de versão	Realização do controle de versão da escrita e dos códigos das aplicações nos cenários.
GitHub	v1.0.13	Hospedagem de código fonte	Armazenamento de código fonte da parte escrita do trabalho e das aplicação web desenvolvidas com base nos cenários de uso estabelecidos.
LaTeX	v5.6.2	Escrita/ formatação	Escrita e formatação do texto.
Linux Mint	v18.3 Sylvia	Sistema Operacional	Sistema operacional para o desenvolvimento dos cenários de uso.
Sublime Text	v3.0	Escrita de código	Edição de textos para escrita do código fonte das aplicações dos cenários de uso.
Ruby	v2.5.0	Linguagem de programação	Linguagem de programação para desenvolvimento das aplicações nos cenários de uso.
RoR	v5.1.5	Desenvolvimento	*Framework* de desenvolvimento web de acordo com o Padrão Arquitetural MVC.
MySQL	v5.7.21	Desenvolvimento	SGBD para desenvolvimento das aplicações nos cenários de uso.
Bootstrap	v4.0.0	Desenvolvimento	*Framework* para desenvolvimento das aplicações no cenários de uso.

5 Catálogo de Segurança

Neste Capítulo, é apresentada uma primeira versão do Catálogo de Segurança. Dado o volume de aspectos associados à Segurança, procurou-se focar nos tópicos mais relevantes, tendo como base a literatura. Dessa forma, o autor está ciente de que o Catalogo de Segurança não acorda todo e qualquer aspecto associado à Segurança. Mas, a intenção de demonstrar que um catálogo construído para especificar RNFs, utilizando uma notação adequada, mesmo que em alto nível de abstração, pode colaborar no entendimento das necessidades inerentes aos aspectos não-funcionais de um software, comumente colocados em segundo plano, ou apenas especificados em documentos, em linguagem natural. Esses documentos, como a Especificação Suplementar (SOMMERVILLE; SAWYER, 1997), carecem de uma notação que permita especificar os RNFs com toda a complexidade envolvida, ou seja, partindo de um alto nível de abstração, mas permitindo acordar operacionalizações, as quais de fato tornarão aquele RNF algo implementável, tratável a nível de código. Para a equipe de desenvolvedores, ter especificadas as operacionalizações pode promover soluções concretas para viabilizar a implementação de aspectos considerados abstratos. Para atingir esse objetivo, o Catálogo de Segurança foi especificado utilizando a notação do NFR *Framework*, já comentada em capítulos anteriores. Essa notação propõe a elaboração de um grafo de interdependências entre os RNFs. Adicionalmente, deixa evidente as principais correlações entre esses RNFs, seus impactos e suas operacionalizações. Conforme justificado no Capítulo 1, dada a relevância do RNF Segurança, têm-se que esse Catálogo de Segurança foca seus esforços na especificação desse RNF.

Para tornar mais compreensível o Catálogo de Segurança, optou-se por apresentá-lo, inicialmente, em dois níveis de abstração. Neste sentido, a seção 5.1.1 apresenta o primeiro nível de abstração. Nesse nível, são acordadas as metas flexíveis (representando os RNFs), seus impactos e suas operacionalizações. Utilizou-se a notação do NFR *Framework*, com a elaboração de um grafo, SIG (*Softgoal Interdependence Graph*). Na seção 5.1.2, é apresentado um mapeamento, procurando correlacionar as metas flexíveis e as camadas do Padrão Arquitetural MVC. Um terceiro nível de abstração também é mapeado e descritro no Capítulo 6, pois para alcançá-lo foi necessário realizar a aplicação do Catálogo de Segurança em cenários. Sendo este, o terceiro nível de abstração entre as operacionalizações e as camadas do Padrão Arquitetural MVC, devido ao fato de envolver aspectos em um nível mais baixo de abstração (implementando de fato as operacionalizações nas camadas *Model-View-Controller*).

5.1 O Catálogo de Segurança

Essa primeira versão do Catálogo de Segurança procura promover, aos especialistas, uma maior compreensão quanto às diferentes necessidades se tratando de Segurança. O Catálogo de Segurança acorda os principais conceitos associados à Segurança, tomando como base o levantamento bibliográfico realizado nessa pesquisa.

Conforme colocado anteriormente, neste capítulo, o Catálogo de Segurança será apresentado em dois níveis de abstração. O primeiro nível, descrito na seção 5.1.1, procura apresentar as metas flexíveis - ou RNFs - associadas à Segurança. O grafo especifica ainda as correlações e os impactos entre essas metas flexíveis, bem como as operacionalizações (no nível mais baixo do grafo). O segundo nível é apresentado na seção 5.1.2, como um mapeamento entre as metas flexíveis e as camadas do Padrão Arquitetural MVC.

5.1.1 Primeiro Nível de Abstração

O primeiro nível de abstração apresenta as metas flexíveis mais genéricas de Segurança de software, apoiando-se, principalmente, na definição de Chung. Essa definição baseia-se em Confidencialidade, Integridade e Disponibilidade. Adicionalmente, essa abstração também se apoia na definição de Segurança da Informação apresentada na ISO 27001.

> "Segurança da informação: preservação da **confidencialidade**, **integridade** e **disponibilidade** da informação; adicionalmente, outras propriedades, tais como **autenticidade**, **responsabilidade**, não repúdio e **confiabilidade**, podem também estar envolvidas."(ISO-27001, 2005, p. 2)

As metas flexíveis foram definidas de acordo com sua relação com Segurança. Observa-se que foram acrescentados, à notação do SIG, alguns valores representados como referências. A intenção é permitir a rastreabilidade de cada aspecto representado no grafo com sua respectiva fonte ou referência teórica. Esse rastreamento permite ao leitor: (i) ter a comprovação de que a especificação do catálogo foi de fato apoiada na literatura, e (ii) aprofundar seus conhecimentos e, caso assim o desejar, consultar as referências para maior esclarecimento sobre cada especificação sugerida no Catálogo de Segurança. A Tabela 9 apresenta os identificadores para a rastreabilidade da referência teórica de uma meta flexível no SIG da Figura 12.

Tabela 9 – Índices de rastreabilidade.

Identificador	Referência
[1]	(CHUNG et al., 2012)
[2]	(BENITTI; RHODEN, 2015)
[3]	(ISO-27001, 2005)
[4]	(AFFLECK; KRISHNA, 2012)
[5]	(BUSCHMANN et al., 1996)

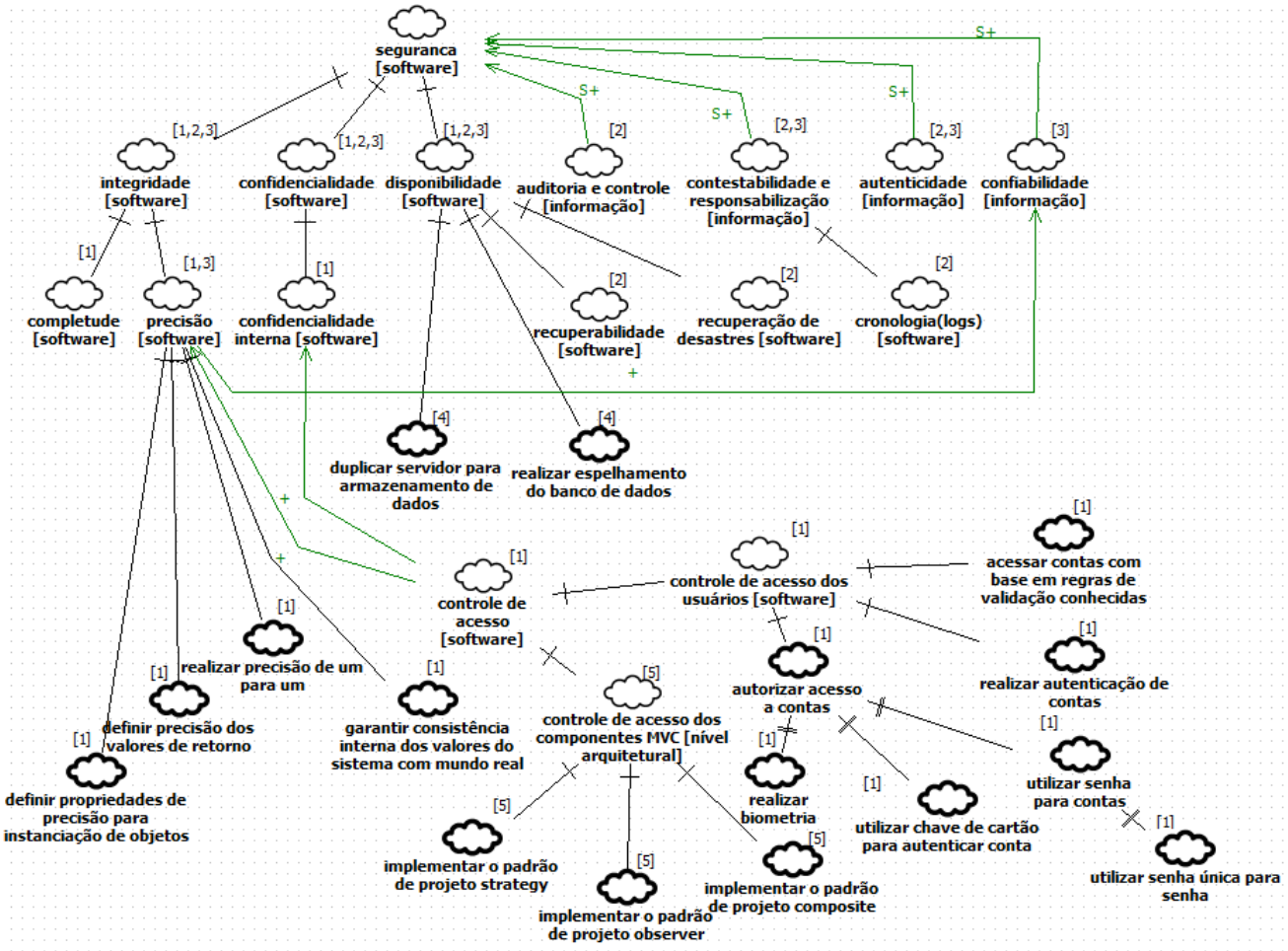


Figura 12 – Catálogo de Segurança.

As definições das metas flexíveis e operacionalizações são:

- **integridade:** proteção contra qualquer tipo de atualização e/ou adulteração não autorizada. Definida na seção 2.6;
  - **precisão:** pode ser entendida como qualquer atributo semântico que fundamenta uma informação. Definida na seção 2.6;
    - ***definir propriedades de precisão para instanciação de objetos:*** garantir que objetos sejam instanciados da maneira correta. Definida na seção 2.6;

    * ***definir precisão dos valores de retorno:*** garantir que os valores retornados pelas operações possuam a precisão pré-estabelecida. Definida na seção 2.6;
    * ***realizar precisão de um para um:*** garantir que um único objeto esteja ligado a uma única entidade do domínio. Definida na seção 2.6;
    * ***garantir consistência interna dos valores do sistema com mundo real:*** garantir que os valores do mundo real sejam correspondentes aos valores do sistema. Definida na seção 2.6;
    * **controle de acesso:** nível mais genérico para autorização de acesso definido por (CHUNG et al., 2012), trata-se das especificações de controle de acesso do software;
        · **controle de acesso dos usuários:** controle de acesso de usuários no software (CHUNG et al., 2012). Para ser satisfeita, depende de um conjunto de operacionalizações: ***autorizar acesso a contas***, ***acessar contas com base em regras de validação conhecidas*** e ***realizar autenticação de contas***;
        · **controle de acesso dos componentes MVC:** trata-se das restrições arquiteturais impostas pelo padrão arquitetural MVC para que as relações internas entre os componentes sejam realizadas (BUSCHMANN et al., 1996). Para que isso ocorra, devem ser implementados três *design patterns*, que são abstraídos no catálogo de Segurança como operacionalizações, sendo elas: (i) ***implementar o padrão de projeto strategy***, (ii) ***implementar o padrão de projeto observer*** e (iii) ***implementar o padrão de projeto composite***;
  – **completude:** garantia que o RNF esteja o mais completo possível. Definida na seção 2.6;

- **confidencialidade:** proteção da informação para evitar que as informações armazenadas ou transmitidas não sejam vistas ou interpretadas por terceiros, sendo somente o usuário principal e o destinatário. Definida na seção 2.6;

  – **confidencialidade interna:** proteção da informação que o acesso deve ser evitado por um público externo. Definida na seção 2.6;

- **disponibilidade:** proteção contra a interrupção do serviço, no momento em que o usuário estiver utilizando a aplicação. Definida na seção 2.6;

  – **recuperabilidade:** intervalo de tempo que o software deverá estar disponível após uma falha (BENITTI; RHODEN, 2015);

  - **recuperação de desastres:** políticas e procedimentos aplicados na recuperação de "desastres" induzidos no software por usuário ou softwares de terceiros (BENITTI; RHODEN, 2015);

  - ***duplicar servidor para armazenamento de dados:*** promove menor chance de tempo onde ocorra a inatividade dos dados. Com a utilização de dois ou mais servidores em funcionamento, a disponibilidade dos dados do software é maior (DATE, 2004)(AFFLECK; KRISHNA, 2012);

  - ***realizar espelhamento do banco de dados:*** compreende-se como duas cópias de um único banco de dados que geralmente reside em máquinas diferentes (DATE, 2004)(AFFLECK; KRISHNA, 2012);

- **auditoria e controle:** especificação dos aspectos que devem ser contemplados para proporcionar a auditoria e o controle (BENITTI; RHODEN, 2015);

- **contestabilidade e responsabilização:** capacidade do software em quantificar as ações e os eventos, com intuito de comprovar sua ocorrência (BENITTI; RHODEN, 2015);

  - **cronologia(logs):** registro de alterações no software (BENITTI; RHODEN, 2015);

- **autenticidade:** capacidade do software em identificar que um objeto ou recurso é o que ele realmente declara ser (BENITTI; RHODEN, 2015), e

- **confiabilidade:** capacidade do software em realizar e manter seu funcionamento quando submetido em circunstâncias de rotina (BENITTI; RHODEN, 2015).

5.1.2 Segundo Nível de Abstração

O segundo nível de abstração procura correlacionar as metas flexíveis mais genéricas e as camadas do Padrão Arquitetural MVC. As Tabelas 10 e 11 apresentam esse mapeamento, considerando diferentes níveis de satisfação para cada meta flexível.

Tabela 10 – Mapeamento das metas flexíveis com as camadas do MVC - Parte 1.

id	Meta flexível	Nível de satisfação	Descrição	Referência	Camada
1	"precisão [software]"	Suficientemente Satisfeita	Garante que um ou mais objetos estejam ligados a uma única entidade do domínio. Além disso, impacta diretamente a *Model* pois é a camada que representa os aspectos do domínio da aplicação. - Atributo de precisão: precisão de um para um.	(CHUNG et al., 2012) (BUSCHMANN et al., 1996)	*model*
2	"precisão [software]"	Suficientemente Satisfeita	Garante que os objetos sejam instanciados da maneira correta dentro da *model*. - Atributo de precisão: Propriedade da precisão.	(CHUNG et al., 2012) (BUSCHMANN et al., 1996)	*Model*
3	"autenticidade [informação]"	Satisfeita Suficientemente	A utilização de *triggers* em banco de dados promove a autenticidade e a verificação da integridade do dado. Além disso, pode ser utilizada para realização de auditoria em tabelas.		
4	"integridade [software]"	Satisfeita Suficientemente			
5	"auditoria e controle [informação]"	Satisfeita Suficientemente		(DATE, 2004)	Banco de Dados
6	"controle de acesso usuário [software]"	Parcialmente Satisfeita	O método de autenticação dos dados retorna a instância do usuário, quando a senha está correta. Esse aspecto é implementado diretamente na *Controller*. Entretanto, possui clara relação com a *View* e com a base de dados. Considera-se "Parcialmente Satisfeita", pois há dependência com a operacionalização "Autorizar Acesso a Contas", a qual contribui para a satisfação dessa meta flexível "controle de acesso [software]".	(FUENTES, 2014)	*Controller*
7	"confiabilidade [informação]"	Suficientemente Satisfeita	Garante que o conjunto de dados a serem armazenados na base de dados estejam relativamente fidedignos. Usa-se "relativamente", pois não há como garantir algo pleno, se tratando de critérios tão abstratos. Nesse caso, "relativamente" sugere que seja o mais fidedigno/confiável possível".	(FUENTES, 2014)	*View*
8	"controle de acesso dos componentes MVC [nível arquitetural]"	Parcialmente Satisfeita	Garante que os componentes da *View*, os quais dependem de outros componentes que se encontram em outras camadas do modelo MVC, reconheçam a necessidade de atualizar as telas, adequando-as às demandas encaminhadas via *Controller* (por exemplo) e em compatibilidade com os dados especificados na *Model*. Considera-se "parcialmente satisfeita", pois há dependência com a implementação do padrão de projeto estrutural *Composite*. A implementação é sugerida como operacionalização que contribue para satisfação dessa meta flexível, "controle de acesso dos componentes MVC [nível arquitetural]."	(BAPTISTELLA, 2011) (BUSCHMANN et al., 1996)	*View*

Tabela 11 – Mapeamento das metas flexíveis com as camadas do MVC - Parte 2.

id	Meta flexível	Nível de satisfação	Descrição	Referência	Camada
8	"controle de acesso dos componentes MVC [nível arquitetural]"	Parcialmente Satisfeita	Garante que os componentes da *View*, os quais dependem de outros componentes que se encontram em outras camadas do modelo MVC, reconheçam a necessidade de atualizar as telas, adequando-as às demandas encaminhadas via *Controller* (por exemplo) e em compatibilidade com os dados especificados na *Model*. Considera-se "Parcialmente Satisfeita", pois há dependência com a implementação do padrão de projeto estrutural *Composite*. A implementação é sugerida como operacionalização que contribue para satisfação dessa meta flexível, "controle de acesso dos componentes MVC [nível arquitetural]."	(BAPTISTELLA, 2011) (BUSCHMANN et al., 1996)	*View*
9	"controle de acesso dos componentes MVC [nível arquitetural]"	Parcialmente Satisfeita	Garante que a *Model* esteja menos acoplada em relação à *View* e à *Controller*, viabilizando essa relação através de boas práticas da Engenharia de Software. Uma dessas práticas é sugerida como operacionalização na Figura 12, apoiando-se no uso de Padrões de Projeto. Considera-se parcialmente satisfeita, pois há dependência com a implementação do padrão de projeto comportamental *Observer*, por exemplo. Essa implementação é sugerida como uma operacionalização possível em atendimento a essa demanda e contribui para a satisfação dessa meta flexível, "controle de acesso dos componentes MVC[nível arquitetural]".	(BAPTISTELLA, 2011) (BUSCHMANN et al., 1996)	*Model*
10	"controle de acesso dos componentes MVC [nível arquitetural]"	Parcialmente Satisfeita	Garante menor acoplamento entre as camadas MVC. Sugere-se que tal aspecto seja apoiado no uso de Padrões de Projeto. Portanto, acredita-se que o padrão de projeto *Strategy*, implementado na *Controller*, permita menor acoplamento entre *View* e *Model*, sendo, de fato, responsabilidade da *Controller* intermediar essa relação utilizando decisões estratégicas. Vale ressaltar que, com o uso desse padrão de projeto decisões podem ser tomadas em tempo de execução, alterando o comportamento do software em função das demandas conhecidas dinamicamente. Dessa forma, a tendência é de maior cumprimento de boas práticas já acordadas no padrão arquitetural MVC. Considera-se parcialmente satisfeita, pois há dependência com a implementação do padrão de projeto comportamental *Strategy*, por exemplo. Essa implementação é sugerida como uma operacionalização possível em atendimento a essa demanda, e contribui para a satisfação dessa meta flexível, "controle de acesso dos componentes MVC[nível arquitetural]".	(BAPTISTELLA, 2011) (BUSCHMANN et al., 1996)	*Controller*

Resumo do Capítulo

Neste capítulo, foi apresentada uma primeira versão do Catálogo de Segurança proposto nessa pesquisa. Adicionalmente, foram apontados os principais impactos e as contribuições das informações do Catálogo de Segurança em relação às camadas do Padrão Arquitetural MVC. Ressalta-se que, o Catálogo foi especificado apoiando-se na literatura, conforme pode ser observado considerando a rastreabilidade, apresentada na seção 5.1.1 evidenciando o primeiro nível de abstração. Nesse nível, foram descritas as metas flexíveis associadas à Segurança. O Catálogo de Segurança especificou as correlações e os impactos entre as metas flexíveis, bem como as operacionalizações.. O segundo nível de abstração foi apresentado na seção 5.1.2, demonstrando o mapeamento entre as metas flexíveis e as camadas do Padrão Arquitetural MVC.

Por fim, cabe mencionar que o Catálogo de Segurança ainda possui mais um nível de abstração, o qual será demonstrado no Capítulo 6, pois só foi possível mapear a correlação entre as operacionalizações e as camadas MVC com a aplicação em cenários. Tal prática permitiu apresentar como os critérios de qualidade podem ser suficientemente tratados e satisfeitos a nível de código.

6 Resultados Obtidos

Neste capítulo, são apresentadas as primeiras aplicações do Catálogo de Segurança e o mapeamento no terceiro nível de abstração (entre operacionalizações e as camadas do Padrão Arquitetural MVC). Para isso, foi utilizado o conceito de *personas*, apresentado na seção 2.7, com o objetivo de demonstrar os cenários nos quais o catálogo pode ser aplicado. Portanto, realizou-se a simulação de cinco cenários para validar a aplicação do Catálogo de Segurança.

O capítulo é dividido, inicialmente, pelos cenários contextualizados dentro de cada *persona*. Os cenários têm a descrição da *persona*, identificação da possível solução (quando necessário descrever para o cenário), e a aplicação do catálogo. Ao final do capítulo, são apresentadas as primeiras impressões sobre a aplicação do catálogo.

6.1 Cenário 1

Heleno, 24 anos, Engenheiro de Requisitos, assumiu a função de elicitar os requisitos não funcionais para um novo projeto na empresa em que trabalha. O projeto será desenvolvido em *Rails* e seu chefe solicitou perfis diferentes de membro e administrador. O sistema solicitado pelo chefe deve, principalmente, ser seguro para autenticação do usuário, bem como, com capacidade de redefinição da senha, monitoramento da quantidade de entradas e, por fim, validação do email do mesmo.

- O problema: mapear os métodos de autenticação relevantes para autenticar o usuário;
- Objetivo: verificar o nível de satisfação de segurança da aplicação, e
- Desafio: analisar os impactos entre os requisitos não funcionais para autenticar os usuários.

6.1.1 Identificar possível solução

O problema principal do usuário é a realização da autenticação do mesmo. O primeiro passo a ser adotado é identificar qual *gem* é a mais adequada. Segundo o *The Ruby Toolbox*, na categoria *Rails Authentication*, o *ranking* de popularidade das três *gems* mais utilizadas para autenticação é (OLSZOWKA, 2018):

1. devise: 38.982.741 *downloads*

2. omniauth: 23.524.509 *downloads*
3. doorkeeper: 7.091.328 *downloads*

Portanto, a *gem* selecionada por Heleno para ser aplicada ao projeto é a *devise*, pois além da sua posição no *ranking*, é a *gem* mais recomendada pela comunidade *Rails*. As vantagens da utilização do *devise* são (i) solução baseada no padrão arquitetural MVC, (ii) criação e a utilização de vários perfis de usuários conectados ao mesmo tempo, e (iii) utilização somente daquilo que é necessário, baseando-se no conceito de modularidade. Essa *gem* possui 10 módulos (RAILS-COMMUNITY, 2019):

1. *Database Authenticatable*: *hashes* que armazenam a senha no banco de dados para serem validadas durante o *login*. A autenticação pode ser realizada via método POST ou HTTP *basic authentication*;
2. *Omniauthable*: suporte adicional ao *omniauth*;
3. *Confirmable*: envio automático de email para confirmação da conta e verificação da confirmação da conta durante o *login*;
4. *Recoverable*: envia instruções de redefinição de senha para o usuário e redefine a senha do usuário;
5. *Registerable*: permite a inscrição de usuários por meio do processo de registros, permitindo a edição e a remoção de suas contas;
6. *Rememberable*: gerenciador de *token* para notificar o usuário da existência de *cookie* salvo;
7. *Trackable*: contador de entradas, mantendo o registro de data, hora e endereço de IP;
8. *Timeoutable*: expira as sessões inativas em um período de tempo especificado;
9. *Validatable*: validações de email e senha, e
10. *Lockable*: bloqueia a conta após exceder o número de tentativas especificado. Portanto, o desbloqueio pode ocorrer via email ou após um período de tempo.

6.1.2 Aplicação do Catálogo de Segurança

A partir do uso do Catálogo de Segurança, foi possível realizar o mapeamento dos módulos apresentados anteriormente. Então, modelou-se a *gem devise* e seus respectivos métodos como operacionalizações no catálogo.

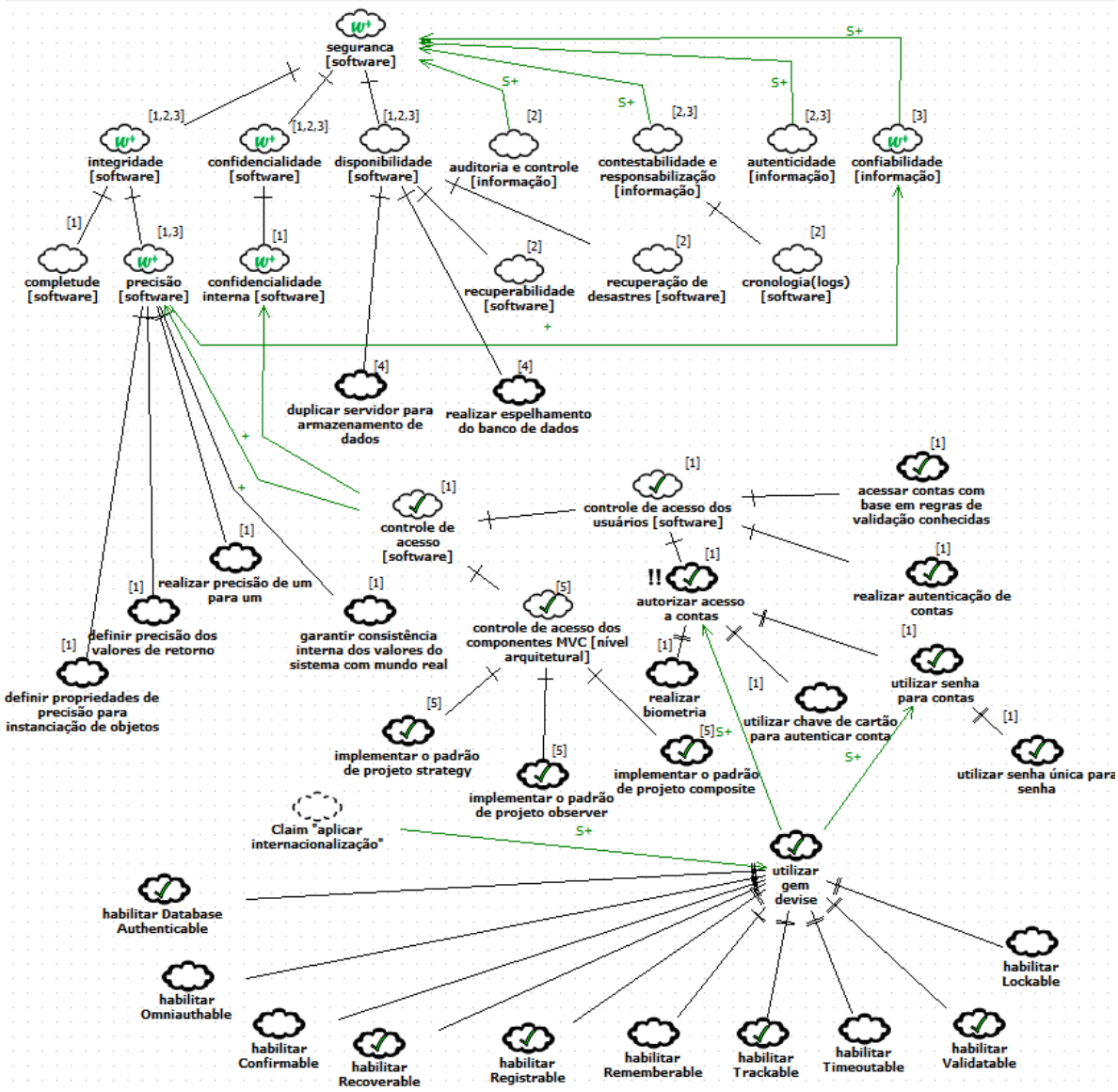


Figura 13 – Operacionalizações para autenticação de usuário utilizando a *gem devise*.

O Catálogo de Segurança aplicado no cenário 1 é apresentado na Figura 13, na qual as operacionalizações foram incrementadas ao catálogo inicial. Respeitando os módulos da *gem devise*, a operacionalização "autorizar acesso a contas" passou a possuir alto grau de prioridade devido ao cenário, pois esta é uma das operacionalizações que impactam diretamente a solução para o problema.

É possível, então, utilizando a notação do NFR *Framework*, realizar a modelagem da *gem devise* como uma operacionalização, representando a existência de alguma contribuição positiva (SOME+) com as operacionalizações "autorizar acesso a contar" e "utilizar senha para contas".

As operacionalizações filhas de "utilizar *gem devise*", e que atendem as necessidades da *persona*, descrita no cenário 1 são:

- "habilitar *Database Authenticatable*": é um dos requisitos mínimos para que ocorra a persistência da senha no banco de dados;
- "habilitar *Recoverable*": capacidade de redefinir a senha do usuário;
- "habilitar *Registrable*": é um dos requisitos mínimos e que permite a inscrição de usuários;
- "habilitar *Trackable*": monitorar a quantidade de entradas do usuário, e
- "habilitar *Validatable*": validar conta via email do usuário.

Portanto, ao habilitar os módulos da *gem devise* dentro da camada *model*, nas classes *Admins* e *Members*, as operacionalizações são satisfeitas. Podemos então, verificar qual nível de satisfação da segurança do software pode ser alcançado com a utilização da *gem devise*.

O trecho de código abaixo apresenta como os módulos podem ser habilitados.

```
1 class Admins < ApplicationRecord
2 devise :database_authenticatable, :registerable,
3 :recoverable, :rememberable, :trackable, :validatable
4 end
```

A implementação dos padrões de projeto *strategy*, *observer* e *composite* já estão suficientemente satisfeitos, pois a aplicação é desenvolvida com o *framework rails* que obedece o padrão MVC (RAILS-COMMUNITY, 2018) e a *gem* que está sendo utilizada também percorre todas as camadas (RAILS-COMMUNITY, 2019).

6.2 Cenário 2

Pedro, 23 anos, Engenheiro de Software, trabalha em empresa privada e possui a necessidade de desenvolver o sistema da ouvidoria da instituição. Tal sistema deverá ser desenvolvido em plataforma web para atender os processos de manifestação (denúncia, reclamação, solicitação, sugestão e elogio) realizados pela ouvidoria. Esse tipo de informação deve ser estritamente confidencial, podendo ou não, o manifestante ser anônimo.

- O problema: sobrecarga de manifestações sem interação com os manifestantes;
- Objetivo: promover a interação anônima ou forma entre a ouvidora e o manifestante, e
- Desafio: realizar a interação anônima entre ouvidora e manifestante, que permita a execução dos processos de auditoria e controle das manifestações.

6.2.1 Identificar possível solução

Uma possível solução é gerar um número de protocolo, durante o registro da manifestação, para que possa ser realizado o acompanhamento. Tal número será utilizado tanto para a manifestação anônima, quanto para a manifestação formal. O manifestante poderá acompanhar o status de sua manifestação através da visão de acompanhamento, na qual ficarão registradas as respostas dadas pela ouvidora.

A Ouvidora, no caso, acessará o sistema de forma diferente, utilizando a área administrativa do sistema através da autenticação de usuário. Ao fazer isso, a mesma estará apta a visualizar todas as manifestações e selecionar a desejada, para que seja possível a redação da resposta para o manifestante.

6.2.2 Aplicação do Catálogo de Segurança

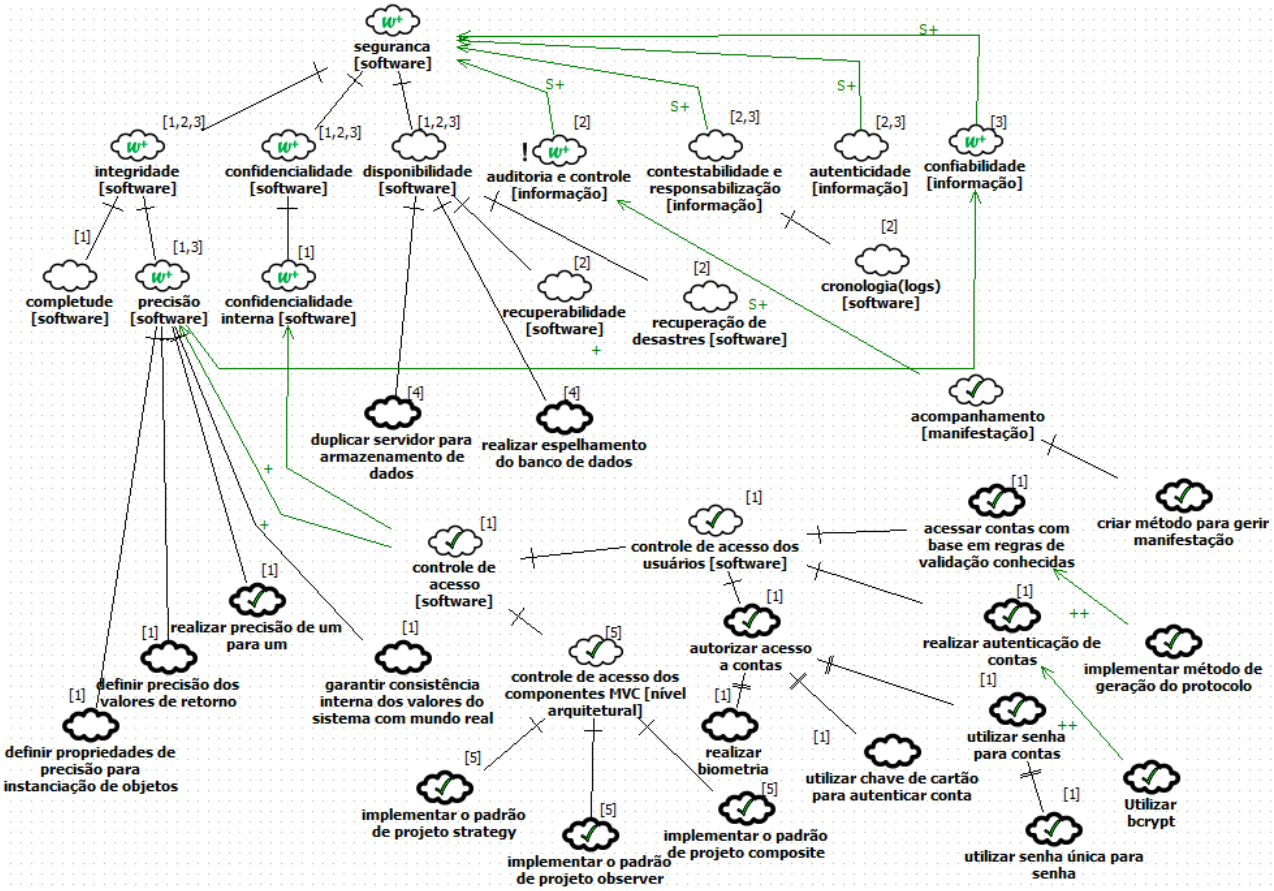


Figura 14 – Catálogo de segurança aplicado ao sistema de ouvidoria.

Na aplicação do Catálogo de Segurança da Figura 14, surgiu a necessidade de focar em “auditoria e controle”, pois o cenário demandava o acompanhamento das manifestações pelos ouvidores. Com isso, foi expandido no Catálogo de Segurança uma nova meta flexível “acompanhamento[manifestações]” e uma operacionalização “Criar método

para gerir manifestação". A operacionalização é definida pelos métodos implementados na *controller* - *ManagerManifestationsController*. Portanto, a meta flexível "auditoria e controle" é suficientemente satisfeita para este contexto.

Para que a meta flexível "controle de acesso dos usuários" seja suficientemente satisfeita, as três operacionalizações que estão no nível mais baixo no Catálogo de Segurança também serão, sendo elas: (i) "implementar método de geração do protocolo": trata-se de um método para gerar os números de protocolos com caracteres (de "a" à "z"), (de "A" à "Z") e (de "0" à "9") composto por dez dígitos; (ii) "utilizar senha única para senha": para cada usuário administrador/ouvidor cadastrado no sistema, é gerada uma única senha para autenticação; e (iii) "utilizar bcrypt": refere-se a uma *gem* que utiliza algoritmo de *hashing* criptográfico tratando um trecho do dado para gerar um *hash*. Considerando que, ao gerar uma senha dessa maneira, trata-se a execução de um processo não reversível, visto que não há como retornar da *hash* para a senha (HALE, 2019).

A operacionalização "realizar precisão de um para um" é suficientemente satisfeita, pois para cada manifestação ocasionada por determinado evento de manifestação do mundo real, existirá uma única manifestação no sistema.

Diante do exposto, observa-se que a segurança do software é parcialmente satisfeita devido a satisfação parcial da integridade e da confidencialidade do mesmo.

6.3 Cenário 3

Milena, 22 anos, Engenheira de Software, possuindo várias atividades no seu dia-a-dia, sentiu a necessidade de organizá-las e para isso, considerando as suas habilidades, decidiu escrever um programa com o objetivo de obter melhor controle sobre as mesmas. Esse programa permite a inserção da atividade com sua descrição, a data de início, a previsão de conclusão e seu status. Consequentemente, Milena optou por fazer seu sistema em *Rails*, e decidiu verificar as relações entre as camadas do sistema com o foco em segurança.

- O problema: desorganização das atividades diárias;
- Objetivo: desenvolvimento de sistema para auxiliar na organização das atividades, e
- Desafio: verificar as relações entre as camadas do sistema com foco em segurança.

6.3.0.1 Aplicação do Catálogo de Segurança

Na descrição da *persona*, houve a necessidade de vincular cada atividade do mundo real com ao menos uma entidade no sistema. Para isso, utilizou-se o comando *scaffold* do

Rails. Tal comando gera a estrutura de arquivos de acordo com o padrão MVC, permitindo obter a *model*, a *controller*, as *views* necessárias (RAILS-COMMUNITY, 2018).

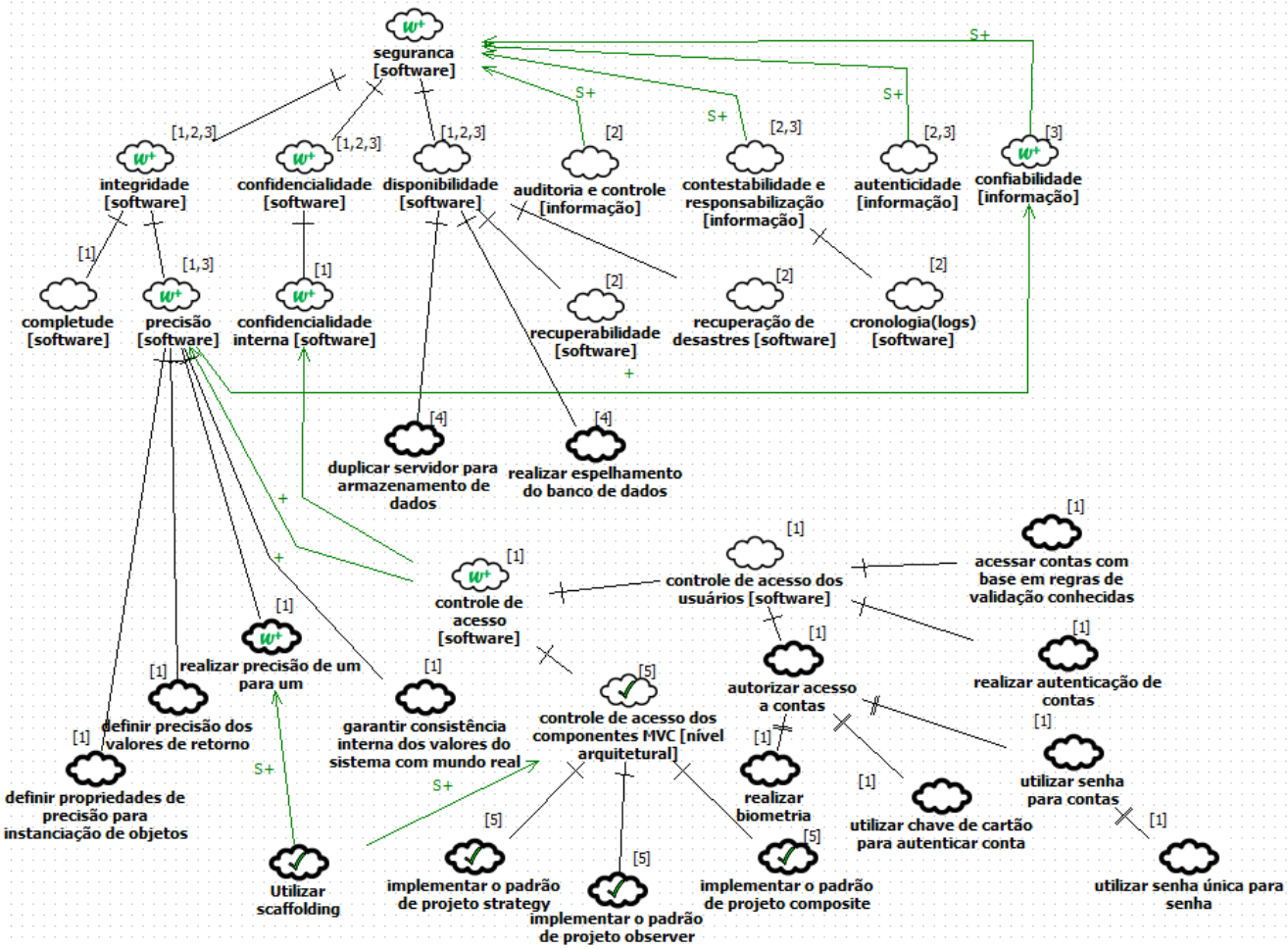


Figura 15 – O *scaffolding* do *Rails* no Catálogo de Segurança.

Como pode ser observado na Figura 15, a operacionalização "utilizar scaffolding" foi estendida ao Catálogo de Segurança, e suficientemente satisfeita devido a implementação ao cenário, satisfazendo parcialmente a operacionalização "realizar precisão de um para um". Portanto as outras metas flexíveis e operacionalizações foram satisfeitas de maneira semelhante aos cenários anteriores.

6.4 Cenário 4

Gustavo, 23 anos, Engenheiro de Requisitos de empresa privada, possui a demanda de analisar os requisitos do sistema de solicitação de documentos digitais via plataforma web. Tal sistema possui arquitetura MVC, e está em constante evolução. Assim sendo, seu chefe solicitou que o mesmo fizesse uma análise dos RNFs com foco na segurança do sistema, devido à emissão de documentos com assinatura digital.

- O problema: alta demanda de solicitação de documentos pelos setores;
- Objetivo: promover a geração de documentos digitais via plataforma web, e
- Desafio: garantir a autenticidade dos documentos solicitados.

6.4.1 Identificar possível solução

A aplicação do Catálogo de Segurança pode ser uma das maneiras de demonstrar a relação entre os RNFs do sistema, principalmente, quando o RNF que está em foco é segurança.

6.4.2 Aplicação do Catálogo de Segurança

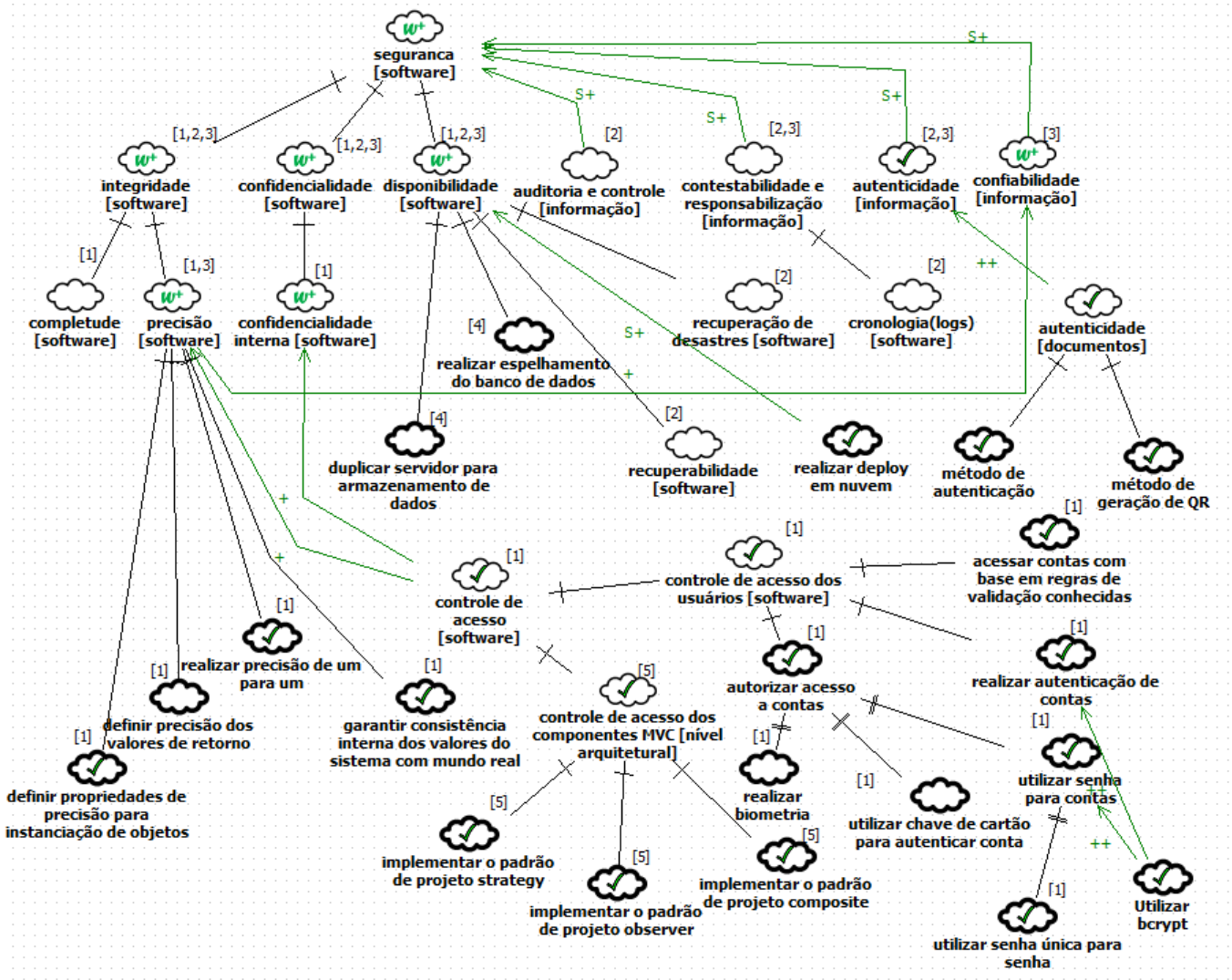


Figura 16 – Catálogo de Segurança aplicado a sistema de geração de documentos digitais.

A aplicação do Catálogo de Segurança apresentado na Figura 16 possui uma extensão devido à utilização de assinatura digital nos documentos, que é modelada como uma

meta flexível que está suficientemente satisfeita, pois depende de duas operacionalizações. As operacionalizações fazem uma relação do tipo AND com a meta flexível e ambas estão suficientemente satisfeitas, sendo elas "método de geração de (*Quick Response* Quick) QR" e o "método de autenticação" que utiliza valores como o identificador do usuário e a *string* do QR. Essas relações satisfazem suficientemente a meta flexível para autenticidade da informação.

O controle de acesso do software é satisfeito devido à relação das metas flexíveis e operacionalizações, da mesma forma que os cenários 6.2 e 6.1 anteriores, que já apresentaram os mesmos relacionamentos, com mesmos níveis de satisfação.

Já a operacionalização "definir propriedades de precisão para instanciação de objetos" é escrita em cada *model*, onde o *Rails* permite escrever os parâmetros para controle de instanciação dos objetos. Neste cenário, foi aplicado que o atributo da classe tem que existir no banco de dados, não podendo colocar vazio ou nulo.

A aplicação está em um servidor da Google, na nuvem, garantindo sua disponibilidade e satisfazendo então, suficientemente, a operacionalização "realizar *deploy* em nuvem".

Semelhante às aplicações dos cenários 6.2 e 6.3, a "precisão de um para um" é suficientemente satisfeita.

6.5 Cenário 5

Chung, 40 anos, Engenheiro de Software, possui em seu quadro de responsabilidades a função de realizar manutenção e evolução de um dos sistemas pelo qual é responsável. Desse modo, percebeu que um de seus sistemas poderia ficar indisponível, caso ocorresse algum problema no servidor de banco de dados. Preocupado com a segurança de seus sistemas, utilizou o Catálogo de Segurança para analisar o impacto que o espelhamento da base de dados causaria na segurança do software como um todo.

- O problema: indisponibilidade do sistema;
- Objetivo: garantir a base disponível para ser utilizada pelo sistema, e
- Desafio: manter a base de dados disponível diante de qualquer interrupção de serviço e que possa impedir acesso ao mesmo.

6.5.1 Aplicação do Catálogo de Segurança

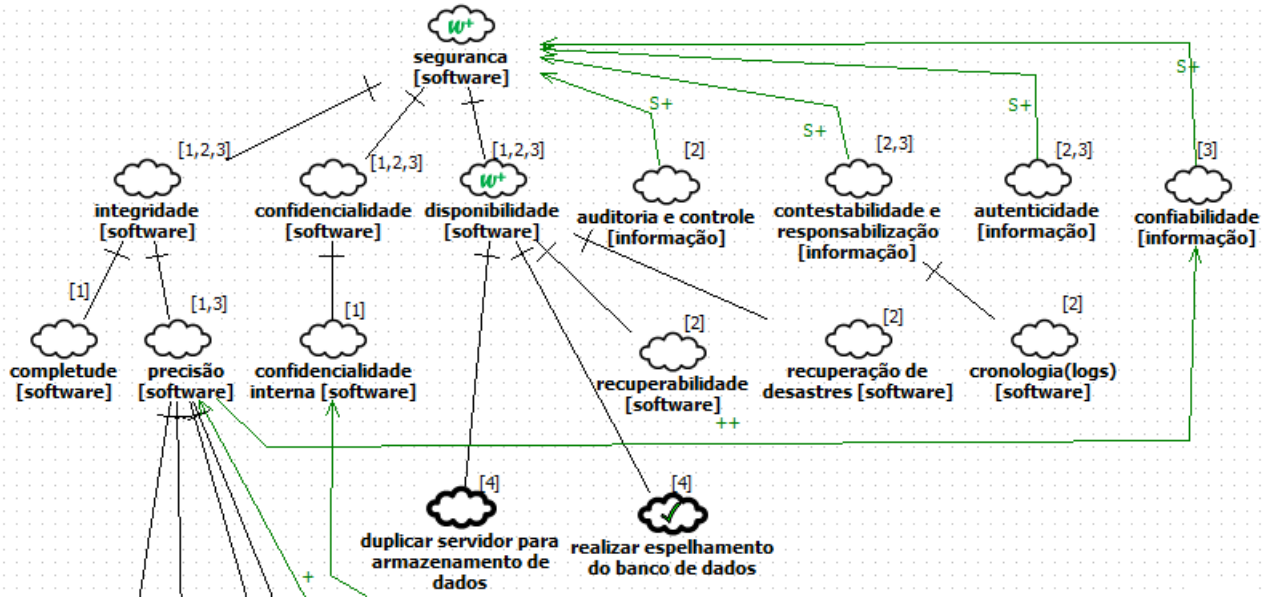


Figura 17 – Catálogo de Segurança com foco em disponibilidade aplicado ao contexto de duplicação de base de dados.

A operacionalização "espelhamento da base de dados" é suficientemente satisfeita, satisfazendo parcialmente a disponibilidade do software. Esse cenário, em especial, foi modelado justamente para demonstrar o impacto de ações realizadas na base de dados na segurança do software, como apresentado na Figura 17.

6.6 Primeira visão do Catálogo de Segurança após as aplicações nos cenários

Ao modelar o Catálogo de Segurança pela primeira vez sem ter realizado nenhuma interação com algum tipo de cenário, obteve-se visão preliminar sobre Segurança devido à subjetividade e ao conjunto de conceitos abstratos presentes na literatura.

Ao demonstrar a aplicação do Catálogo de Segurança nos cenários, observou-se que o mesmo permite, não somente a visão preliminar da segurança de software, mas também de segurança da informação.

Como o Catálogo de Segurança foi aplicado em cenários onde a ferramenta de apoio ao desenvolvimento era o *Rails*, foi possível abstraí-lo para aplicações em *Rails*, demonstrando então, a adaptabilidade do mesmo, como pode ser visualizado na Figura 18.

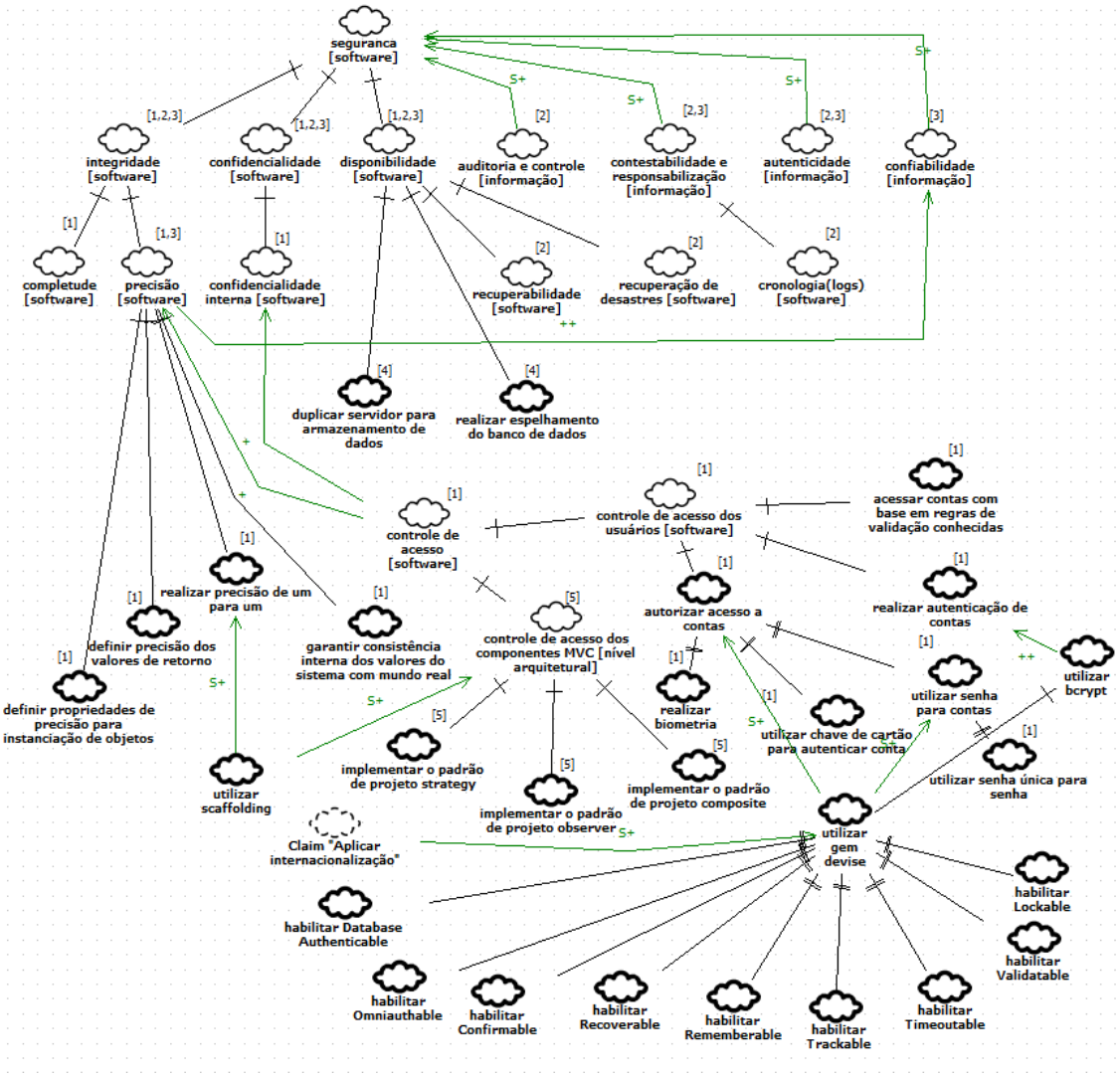


Figura 18 – Versão extendida do Catálogo de Segurança focada em projetos *Rails*.

Entretanto, durante a validação do Catálogo de Segurança, foram utilizados projetos em cenários reais e abstrações de possíveis usuários e cenários, tornando-o mais abrangente, caso a tecnologia utilizada seja o *Rails*. Como um exemplo de evolução do Catálogo de Segurança e as evidências comprovadas da relação entre as operacionalizações e as camadas do Padrão Arquitetural MVC, é possível correlacionar, com fins de demonstração, o Catálogo de Segurança com um diagrama de componentes do padrão MVC, onde as operacionalizações estão dentro do componente ao qual possui relacionamento, como pode ser visualizado na Figura 19.

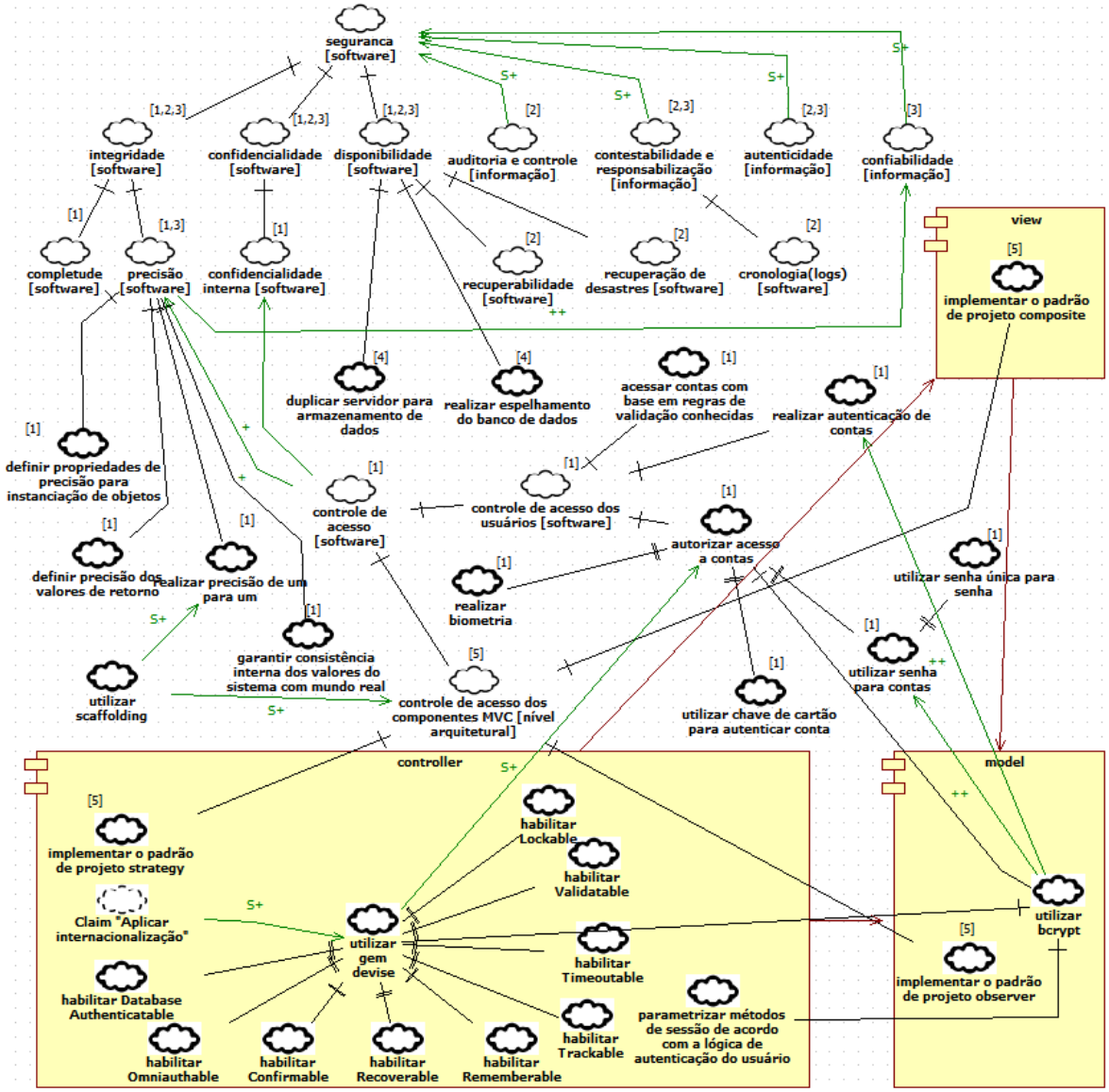


Figura 19 – Mapeamento das metas flexíveis e operacionalizações com as camadas do MVC.

Resumo do Capítulo

Neste capítulo, foram apresentadas as primeiras aplicações do Catálogo de Segurança, realizadas através de cinco cenários, onde foi possível demonstrar a aplicabilidade e a adaptabilidade do Catálogo de Segurança, de acordo com o contexto. A partir da seção 6.6, foi possível evidenciar a evolução do Catálogo de Segurança para as aplicações em projetos em *Rails* e, por fim, o mapeamento em terceiro nível de abstração, sendo viável identificar as operacionalizações relacionadas com as camadas do Padrão Arquitetural MVC para projetos em *Rails*.

7 Conclusão

Dada a complexidade do tema abordado, em especial por procurar contribuir a partir de critérios de qualidade que são intrinsecamente abstratos e subjetivos, observou-se que o Catálogo de Segurança reflete uma visão preliminar sobre Segurança. Espera-se ainda, com o Catálogo de Segurança, que as contribuições desse trabalho possam ser compreendidas, ajustadas e aplicadas para e em outros cenários de uso. A ideia foi demonstrar como é possível tratar aspectos subjetivos, como os RNFs, permitindo satisfazê-los com a implementação de operacionalizações. Ou seja, demonstrar os elos de ligação entre um RNF e algo mais viável de ser concretizado em termos de arquitetura e código. Além disso, tudo fica anotado, utilizando uma notação em diferentes níveis de abstração, e deixando claro os impactos e as correlações entre esses níveis.

Uma vez evidenciadas as correlações entre os níveis de abstração, dos requisitos ao código, procurou-se conferir uma ligação entre algo muito abstrato (requisitos), com algo muito concreto (código). Esses níveis de abstração são comumente vistos como muito distantes, envolvendo e demandando - muitas vezes - equipes técnicas diferentes. Tal distanciamento, pode incorrer no não entendimento, ou não atendimento correto, ou ainda no próprio esquecimento de alguns desses aspectos subjetivos. Essa prática, conforme acordado no referencial teórico desse trabalho, o qual foi apoiado na literatura da área, pode levar - e recorrentemente leva - a insucessos nos projetos de software.

A Tabela 12 apresenta, de acordo com os objetivos específicos do trabalho, os níveis de satisfação e os motivos pelos quais os mesmos foram atendidos.

Tabela 12 – Níveis de satisfação dos objetivos específicos.

Objetivo	**Nível de satisfação**	**Motivo**
Investigar na literatura formas de lidar com o RNF Segurança em aplicações Web desenvolvidas utilizando o MVC.	Atendido	Atendido com base em estudos realizados na área de interesse, com os quais referências foram levantadas e sustentam a necessidade de se considerar o RNF Segurança bem como seus impactos em aplicações Web, desenvolvidas com base no Padrão Arquitetural MVC.
Investigar na literatura os RNF associados à segurança, e identificar o impacto e as interdependências entre eles.	Atendido	Atendido pela existência de fontes que evidenciam a relação entre os RNFs de segurança em diferentes níveis de abstração.
Elaborar SIG	Atendido	Atendido, pois tem-se a primeira versão do catálogo elaborado com sucesso.
Realizar correspondência entre o Catálogo de Segurança e as camadas do Padrão Arquitetural MVC	Atendido	Atendido, pois o presente trabalho demonstrou a existência e os impactos das relações entre os RNFs que geram impacto na Segurança do Software em três níveis de abstração
Elaborar cenários e desenvolver aplicações web exemplo, no padrão arquitetural MVC, orientando-se pelo Catálogo de Segurança	Atendido	Atendido, pois, para validação do Catálogo de Segurança, o mesmo foi aplicado em cinco cenários, que podem ser vistos como cenários de uso, em alguns casos, e estudo de casos, em outros.

Trabalhos Futuros

Muitos trabalhos futuros podem ser desenhados a partir das contribuições e iniciativas desse trabalho. Dentre as ideias para projetos futuros, destacam-se:

Podem ser realizadas correlações desse catálogo, focado em Segurança, com outros catálogo, em diferentes critérios de qualidade. De acordo com o levantamento realizado nesse trabalho, têm-se que os primeiros critérios candidatos à investigação, dada à relevância dos mesmos na literatura, são Desempenho e Usabilidade. Anotar os impactos entre esses catálogos e o presente catálogo poderá conferir uma série de novas colocações. Por exemplo, é claro que se segurança for respeitada ao extremo, isso impactará em aspectos de usabilidade. Solicitar nome do pai, nome da mãe, digital e data de nascimento, conferem maior segurança em um caixa eletrônico, mas comprometem aspectos de usabilidade. Essas e outras conclusões poderiam ser mapeadas, acordadas, permitindo estudos mais aprofundados desses aspectos subjetivos no desenvolvimento de software.

O Catálogo de Segurança, apresentado neste trabalho, também pode evoluir a níveis cada vez mais baixos de abstração, sendo capaz de conferir respostas a questionamentos bem específicos, como: “qual método de criptografia pode ser mais eficiente para uma determinada operacionalização e que está diretamente ligado à segurança do software?”, ou ainda ser expandido para indagações na área de segurança da informação como auditoria e controle, contestabilidade e responsabilização, autenticidade ou confiabilidade.

Por fim, poderiam ser mencionados suportes computacionais que permitissem que o catálogo fosse disponibilizado para a comunidade interessada, conferindo - inclusive - mecanismos de compartilhamento de conhecimento e possibilidades de evoluções colaborativas do catálogo.

Referências

AFFLECK, A.; KRISHNA, A. Supporting quantitative reasoning of non-functional requirements: A process-oriented approach. In: IEEE PRESS. *Proceedings of the International Conference on Software and System Process*. [S.l.], 2012. p. 88–92. Citado 5 vezes nas páginas 15, 36, 37, 63 e 65.

ASTAH. *Astah Quick Start Guide*. 2017. Disponível em: <http://astah.net/tutorial/pro/quick-start-guide>. Citado na página 57.

BAPTISTELLA, A. J. Abordando a arquitetura mvc, e design patterns: Observer, composite, strategy. *Linha de Código,[Online]. Available: http://www. linhadecodigo. com. br/artigo/2367/abordando-a-arquitetura-mvc-e-designpatterns-observer-composite-strategy. aspx.[Acesso em 27 Maio 2016]*, 2011. Citado 2 vezes nas páginas 66 e 67.

BENITTI, F. B. V.; RHODEN, J. S. Uma taxonomia unificada para requisitos não funcionais. *Revista Electronica de Sistemas de Informaçao*, Faculdade Cenecista de Campo Largo-FACECLA, v. 14, n. 3, p. 1, 2015. Citado 3 vezes nas páginas 63, 64 e 65.

BIZAGI. *Bizagi BPMN Modeler*. 2018. Disponível em: <https://www.bizagi.com/pt/produtos/bpm-suite/modeler>. Citado na página 58.

BOOTSTRAP. *Bootstrap - The most popular HTML, CSS, and JS library in the world.* 2018. Disponível em: <https://getbootstrap.com/>. Citado na página 59.

BUSCHMANN, F. et al. A system of patterns: Pattern-oriented software architecture. Wiley New York, 1996. Citado 12 vezes nas páginas 15, 17, 23, 24, 39, 40, 41, 43, 63, 64, 66 e 67.

CHUNG, L.; LEITE, J. do P. On non-functional requirements in software engineering. *Conceptual modeling: Foundations and applications*, Springer, p. 363–379, 2009. Citado na página 24.

CHUNG, L. et al. *Non-functional requirements in software engineering*. [S.l.]: Springer Science & Business Media, 2012. v. 5. Citado 18 vezes nas páginas 15, 23, 24, 30, 31, 33, 34, 35, 36, 43, 44, 45, 46, 47, 53, 63, 64 e 66.

CINTRA, C. C. A implementação de um processo de engenharia de requisitos baseado no processo unificado da rational (rup) alcançando nível 3 de maturidade da integração de modelos de capacidade e maturidade (cmmi) incluindo a utilização de práticas de métodos ágeis. 2006. Citado 2 vezes nas páginas 37 e 38.

CYSNEIROS, L. M.; LEITE, J. Definindo requisitos não funcionais. *XI Simpósio Brasileiro de Engenharia de Software. Fortaleza, CE*, p. 33, 1997. Citado 2 vezes nas páginas 32 e 33.

DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004. Citado 2 vezes nas páginas 65 e 66.

DURELLI, V. H.; VIANA, M. C.; PENTEADO, R. A. Uma proposta de reuso de interface gráfica com o usuário baseada no padrão arquitetural mvc. *Simpósio Brasileiro de Sistemas de Informação, Rio de Janeiro–RJ, anais SBSI*, p. 48–59, 2008. Citado 3 vezes nas páginas 15, 42 e 43.

ECKHARDT, J.; VOGELSANG, A.; FERNÁNDEZ, D. M. Are non-functional requirements really non-functional?: an investigation of non-functional requirements in practice. In: ACM. *Proceedings of the 38th International Conference on Software Engineering*. [S.l.], 2016. p. 832–842. Citado 3 vezes nas páginas 23, 26 e 33.

FONSECA, J. J. S. Metodologia da pesquisa científica. 2002. Citado na página 50.

FORD, D. et al. Characterizing software engineering work with personas based on knowledge worker actions. In: IEEE. *Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on*. [S.l.], 2017. p. 394–403. Citado na página 47.

FUENTES, V. B. *Ruby on Rails: Coloque sua aplicação web nos trilhos*. [S.l.]: Editora Casa do Código, 2014. Citado na página 66.

GERHARDT, T. E.; SILVEIRA, D. T. Métodos de pesquisa. coordenado pela universidade aberta do brasil–uab/ufrgs e pelo curso de graduação tecnológica–planejamento e gestão para o desenvolvimento rural da sead/ufrgs. *Porto Alegre: Editora da UFRGS*, v. 2, n. 0, p. 0, 2009. Citado 2 vezes nas páginas 49 e 50.

GIL, A. C. Como elaborar projetos de pesquisa. *São Paulo*, v. 5, n. 61, p. 16–17, 2002. Citado na página 50.

GIT. *About Git*. 2017. Disponível em: <https://git-scm.com/about>. Citado na página 58.

GITHUB. *GitHub*. 2017. Disponível em: <https://github.com/>. Citado na página 58.

GORDIEIEV, O. et al. Evolution of software quality models in context of the standard iso 25010. In: SPRINGER. *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland*. [S.l.], 2014. p. 223–232. Citado na página 37.

HALE, C. *bcrypt-ruby*. 2019. Disponível em: <https://github.com/codahale/bcrypt-ruby>. Citado na página 74.

HALEY, C. B. et al. A framework for security requirements engineering. In: ACM. *Proceedings of the 2006 international workshop on Software engineering for secure systems*. [S.l.], 2006. p. 35–42. Citado na página 44.

HORKOFF, J. et al. Goal-oriented requirements engineering: A systematic literature map. In: IEEE. *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. [S.l.], 2016. p. 106–115. Citado na página 24.

HORKOFF, J.; YU, E. *iStarQuickGuide*. 2006. Disponível em: <http://istar.rwth-aachen.de/tiki-index.php?page=iStarQuickGuide>. Citado 2 vezes nas páginas 24 e 58.

ISO-27001. Information technology–security techniques–information security management systems–requirements. 2005. Citado 3 vezes nas páginas 43, 62 e 63.

ISO/IEC-25010. *ISO-IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models*. [S.l.]: ISO, 2011. Citado 2 vezes nas páginas 17 e 33.

JACKSON, M. Problem frames and software engineering. *Information and Software Technology*, Elsevier, v. 47, n. 14, p. 903–912, 2005. Citado na página 58.

JAILIA, M. et al. Behavior of mvc (model view controller) based web application developed in php and. net framework. In: IEEE. *ICT in Business Industry & Government (ICTBIG), International Conference on*. [S.l.], 2016. p. 1–5. Citado na página 23.

KEELE, S. Guidelines for performing systematic literature reviews in software engineering. In: *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. [S.l.]: sn, 2007. Citado na página 25.

KOTONYA, G.; SOMMERVILLE, I. *Requirements engineering: processes and techniques*. [S.l.]: Wiley Publishing, 1998. Citado na página 30.

LAMSWEERDE, A. V. Goal-oriented requirements engineering: A guided tour. In: IEEE. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. [S.l.], 2001. p. 249–262. Citado 3 vezes nas páginas 31, 32 e 58.

LATEX. *LaTeX - A document preparation system*. 2017. Disponível em: <https://www.latex-project.org/>. Citado na página 59.

LINUX-MINT-COMMUNITY. *Linux Mint User Guide*. 2018. Disponível em: <https://linuxmint-installation-guide.readthedocs.io/pt_BR/latest/>. Citado na página 59.

MAIRIZA, D.; ZOWGHI, D.; NURMULIANI, N. An investigation into the notion of non-functional requirements. In: ACM. *Proceedings of the 2010 ACM Symposium on Applied Computing*. [S.l.], 2010. p. 311–317. Citado na página 23.

MAZZOTTI, A. J. A. O planejamento de pesquisas qualitativas em educação. *Cadernos de pesquisa*, Fundaçao Carlos Chagas, n. 77, p. 53–61, 1991. Citado na página 49.

MEIRA, S. *Palestra Silvio Meira CIn/UFPE: An Agenda for the Future of Software [Engineering in Brazil]*. 2015. Disponível em: <https://www.youtube.com/watch?v=eujhiejLL7c>. Citado na página 31.

NBR-ISO/IEC-9126-1. *NBR ISO/IEC 9126-1:Engenharia de Software - Qualidade de produto. Parte 1: Modelo de qualidade*. [S.l.]: NBR, 2003. Citado 2 vezes nas páginas 37 e 38.

OLSZOWKA, C. *Ruby-toolbox*. 2018. Disponível em: <https://www.ruby-toolbox.com/search?utf8=%E2%9C%93&q=authentication>. Citado na página 69.

ORACLE. *Visão Geral - MySQL*. 2018. Disponível em: <https://www.oracle.com/br/mysql/index.html>. Citado na página 59.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, ACM, v. 17, n. 4, p. 40–52, 1992. Citado na página 38.

PRESSMAN, R. S. Engenharia de software: uma abordagem profissional. 7ª edição. *Ed: McGraw Hill*, 2011. Citado 2 vezes nas páginas 30 e 37.

PRIKLADNICKI, R.; WILLI, R.; MILANI, F. *Métodos ágeis para desenvolvimento de software*. [S.l.]: Bookman Editora, 2014. Citado na página 51.

RAILS-COMMUNITY. *Getting Started with Rails*. 2018. Disponível em: <https://guides.rubyonrails.org/v3.2.8/getting_started.html>. Citado 2 vezes nas páginas 72 e 75.

RAILS-COMMUNITY. *Devise Documentation*. 2019. Disponível em: <https://github.com/plataformatec/devise>. Citado 2 vezes nas páginas 70 e 72.

RAILSCOMMUNITY. *Imagine what you could build if you learned Ruby on Rails*. 2018. Disponível em: <http://rubyonrails.org/>. Citado na página 59.

RAMOS, E. de S. Elaboração de uma persona para o profissional de análise de requisitos que pratica ux/ucd/ihc baseado em dados estatísticos provenientes de pesquisas no contexto brasileiro. EATI-Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação - Fundação Getúlio Vargas (FGV), 2013. Citado na página 47.

RE-TOOLS. *RE-Tools: A Multi-notational Requirements Modeling Toolkit*. 2017. Disponível em: <http://www.utdallas.edu/~supakkul/tools/RE-Tools/>. Citado na página 58.

REIS, B.; MOTA, J. C.; OLIVEIRA, P. P. B. de. *Classificação da Informação*. 2001. Citado na página 45.

RUBYCOMMUNITY. *Sobre o Ruby*. 2018. Disponível em: <https://www.ruby-lang.org/pt/about/>. Citado na página 59.

SCHNEIDEWIND, N. Standard for a software quality metrics methodology. *Soft. Eng. Standards Subcommittee of the IEEE*, 1990. Citado na página 26.

SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1. Citado na página 51.

SHAW, M.; GARLAN, D. *Software architecture: perspectives on an emerging discipline*. [S.l.]: Prentice Hall Englewood Cliffs, 1996. v. 1. Citado na página 39.

SILVA, R. F.; CUNHA, J. A. Arquitetura de segurança em aplicações baseadas em web services. *HOLOS*, v. 3, p. 15–24, 2007. Citado na página 44.

SOEGAARD, M.; DAM, R. F. The encyclopedia of human-computer interaction. *The Encyclopedia of Human-Computer Interaction*, The Interaction Design Foundation, 2012. Citado na página 47.

SOMMERVILLE, I. et al. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2003. v. 6. Citado na página 29.

SOMMERVILLE, I.; SAWYER, P. *Requirements engineering: a good practice guide*. [S.l.]: John Wiley & Sons, Inc., 1997. Citado 3 vezes nas páginas 23, 32 e 61.

STARUML. *About StarUML*. 2017. Disponível em: <http://staruml.sourceforge.net/v1/about.php>. Citado na página 57.

SUBLIME-TEXT. *About Sublime*. 2018. Disponível em: <https://www.sublimetext.com/>. Citado na página 59.

SULLIVAN, B.; LIU, V. *Web application security, a beginner's guide*. [S.l.]: McGraw-Hill Education Group, 2011. Citado na página 44.

SUPAKKUL, S.; CHUNG, L. The re-tools: A multi-notational requirements modeling toolkit. In: IEEE. *Requirements Engineering Conference (RE), 2012 20th IEEE International*. [S.l.], 2012. p. 333–334. Citado na página 58.

UMAR, M.; KHAN, N. A. Analyzing non-functional requirements (nfrs) for software development. In: IEEE. *Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on*. [S.l.], 2011. p. 675–678. Citado 2 vezes nas páginas 24 e 37.

WHITE, S. A. Introduction to bpmn. *Ibm Cooperation*, v. 2, n. 0, p. 0, 2004. Citado na página 57.

WIEGERS, K.; BEATTY, J. *Software requirements*. [S.l.]: Pearson Education, 2013. Citado na página 24.

XMIND.LTD. *About XMind*. 2017. Disponível em: <http://www.xmind.net/about/>. Citado 2 vezes nas páginas 57 e 58.