UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DEPARTAMENTO ACADÊMICO DA COMPUTAÇÃO – DACOM
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

IVAN DAL VESCO

REFATORAÇÃO E INCREMENTAÇÃO DE FERRAMENTA WEB PARA A ELABORAÇÃO DE REVISÃO SISTEMÁTICA

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

IVAN DAL VESCO

REFATORAÇÃO E INCREMENTAÇÃO DE FERRAMENTA WEB PARA A ELABORAÇÃO DE REVISÃO SISTEMÁTICA

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof Alessandra Bortoletto Garbelotti Hoffmman.

Co-orientador: Prof Juliano Rodrigo Lamb.

MEDIANEIRA

2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Refatoração e Incrementação de Ferramenta Web para a Elaboração de Revisão Sistemática

Por

Ivan Dal Vesco

Este Trabalho de Diplomação (TD) foi apresentado às 10:20 h do dia 17 de novembro de 2015 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Câmpus* Medianeira. O acadêmico foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Me. Alessandra Garbelotti Hoffmman
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Me. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Co-Orientador)

Prof. Me Glória Patricia Lopez Sepulveda
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Me Jorge Aikes
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

VESCO, Ivan dal. REFATORAÇÃO E INCREMENTAÇÃO DE FERRAMENTA WEB PARA A ELABORAÇÃO DE REVISÃO SISTEMÁTICA. 75 f. Trabalho de Diplomação - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Medianeira, 2015.

Quando se desenvolve um *software*, por vezes são omitidos passos, etapas ou soluções já desenvolvidas que auxiliam o desenvolvimento de um produto final com qualidade e flexibilidade. Desta maneira, quando algum *software* é desenvolvido de modo inadequado, por vezes voltar às etapas e corrigir os problemas não é o suficiente, sendo necessário reiniciar todo o processo de desenvolvimento do *software*, revendo e reconstruindo todas as etapas. As falhas mais comuns estão relacionadas à etapa de análise, esta que é responsável pela abstração, identificação dos pontos-chave que se buscam as soluções e a aplicação de conceitos e técnicas para a modelagem de um produto que atenda às expectativas e possua flexibilidade, para que futuramente quando houverem mudanças nas necessidades de quem utiliza este produto, o mesmo possua condições de acompanhar esta demanda. O tema proposto consiste em reestruturar um projeto já iniciado, refatorando desde às etapas de análise, utilizando a *unified modeling language* para a documentação, aplicando técnicas de programação orientada a objetos, conceitos de padrões de projeto e princípios de projeto, projetando a importância de uma análise adequada e fundamentada para o desenvolvimento futuro do projeto. O projeto tem como objetivo o desenvolvimento de uma ferramenta que disponibilize o processo de revisão bibliogrática sistemática em ambiente *web*, oferecendo a colaboratividade e integração como vantagens do produto final.

Palavras-chave: Análise, Padrões de Projeto, Colaboratividade.

ABSTRACT

VESCO, Ivan dal. REFACTORING AND INCREASE WEB TOOL FOR DEVELOPING SISTEMATIC REVIEW. 75 f. Graduation's Labor- Degree in Technology Analisys and System Development, Federal Technological University of Paraná, Medianeira, 2015.

When developing software sometimes footsteps, steps and solutions are omitted, solutions these which quality and flexibility to the final product development . That way, when some software is developed improperly, sometimes isn't enough back to the steps and correct them, and restart the entire software development process, reviewing and rebuilding all stages. The most common faults related to the analysis stage, this one which is responsible for abstraction, identification of key points which seek solutions and the application of concepts and techniques for modeling a product that meets the expectations and have flexibility to that in the future when there are changes in the needs of those who use this product, this has able to follow this demand. The proposed theme is to structure the project initiated by Ferreira (2014), reshaping the project from the analysis steps, using UML for documentation, applying OOP techniques, design patterns concepts and design principles, demonstrating the importance of proper and reasoned analysis for the future development of the project. The project aims to develop a tool that provides the process of systematic literature review in a web environment, offering colaborativity and integration as the final product advantages.

Keywords: Analysis, Design Patterns, Colaborativity.

LISTA DE ABREVIAÇÕES

API	Application Programming Interface
CSS	Cascading Style Sheet
GoF	Gang of Four
HTML	Hyper Text Markup Language
JVM	Java Virtual Machine
MVC	Model View Controller
POO	Programação Orientada a Objetos
UFSCAR	Universidade Federal de São Carlos
UML	Unified Modeling Language

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Segundo Conforto, Amaral e Silva (2011), a revisão bibliográfica sistemática é um método científico para busca e análise de artigos de uma determinada área da ciência.

A necessidade de se conduzir pesquisas que ofereçam resultados confiáveis foi o principal fator que culminou no desenvolvimento da revisão sistemática literária, este método surgiu inicialmente na área de saúde na qual supria a necessidade de aquisição de informações a respeito de tomadas de decisão de modo que a confiabilidade dos estudos fosse fator primordial, pois destes estudos baseavam-se a indicação de tratamentos eficazes.

> "Esta metodologia fundamenta-se no Movimento de Pesquisa Baseada em Evidências, que emergiu do aumento da produção científica mundial, do crescente número de intervenções, tecnologias, medicamentos e terapias na área de saúde, e da necessidade de validar os resultados obtidos a partir de vários estudos sobre determinada questão, a fim de subsidiar a tomada de decisão." (Lopes; 2008, p.772, apud EVANS, 2001).

Observa-se então a importância da revisão sistemática literária, não só para a área da saúde, mas para qualquer outra área que exija estudos bibliográficos confiáveis sobre determinado tema que auxiliem às tomadas de decisão ou fundamentem outros estudos.

Para auxiliar pesquisadores no desenvolvimento da revisão sistemática, há diversas ferramentas oferecidas no mercado, como, por exemplo, a *Start* desenvolvida pelo LAPES (Laboratório de Pesquisa em Engenharia de Software) da Universidade Federal de São Carlos (UFSCAR), que permite a elaboração de uma revisão sistemática por meio de uma ferramenta *desktop*, permitindo asssim apenas sua utilização individual, não permitindo o uso de forma colaborativa, com limitações relacionadas às buscas que não são realizadas de forma automática.

Pela ausência de uma ferramenta colaborativa que permita a integração de esforço entre pesquisadores de diferentes lugares e a automatização dos processos de busca e integração iniciou-se então o desenvolvimento de uma ferramenta que pudesse atingir estes requisitos e oferecer uma maneira de expandir, integrar e agilizar o processo de revisão bibliográfica sistemática.

Dando continuidade ao desenvolvimento desta ferramenta, identificou-se a impossibilidade de agregar funcionalidades ao modelo desenvolvido, pela ausência de um modelo de classes flexível, que seguisse os preceitos de boas práticas, técnicas de programação orientada a objetos (POO) e a utilização de soluções já desenvolvidas (padrões de projeto) que vem de encontro à algumas das funcionalides da ferramenta, de modo que, tornou-se necessário refatorar toda a fundamentação do projeto, focando na elaboração de um modelo com flexibilidade e documentação adequados para que o projeto possa ser desenvolvido e incrementado sem preocupar-se com os impactos gerados por estas alterações.

1.1 OBJETIVO GERAL

Refatorar, documentar e aplicar técnicas de POO e conceitos de padrões de projeto no projeto de desenvolvimento de uma ferramenta *Web* de revisão bibliográfica sistemática.

1.2 OBJETIVOS ESPECÍFICOS

- Analisar a aplicação desenvolvida;
- Remodelar e incluir funcionalidades ao trabalho de Ferreira(2014);
- Readequar a etapa de análise do projeto;
- Estudar e aplicar ténicas de orientação a objetos e padrões de projeto;
- Estudar linguagens já utilizadas para o desenvolvimento.

1.3 JUSTIFICATIVA

Apesar de todos os estudos na área de *softwares*, os processos de desenvolvimento de revisões sistemáticas ainda baseiam-se em métodos manuais para a construção de suas fases e etapas, de forma local, utilizando ferramentas em ambiente *desktop*, de forma não colaborativa. Tendo em vista que existem apenas ferramentas para ambiente *desktop* para a elaboração destas revisões, sendo em sua maioria ferramentas proprietárias, é identificada a necessidade de desenvolvimento de uma ferramenta em ambiente *Web* que permita a elaboração do processo de revisão sistemática de modo ágil, integrado, colaborativo e utilizando mecanismos de busca integrados. Oferecendo assim a portabilidade necessária para que pesquisadores de diferentes lugares consigam realizar este processo de forma unificada.

Para que esta ferramenta possua meios de crescer e desenvolver-se adequadamente é proposta a readequação de sua primeira versão desenvolvida por Ferreira (2014), de modo que o modelo desenvolvido inicialmente não consegue atender os requisitos para dar continuidade ao projeto, impossibilitando a incrementação de funcionalidades, pela ausência de uma construção e documentação de seus modelos que seguissem os paradigmas de princípios de projeto, técnicas de POO ou ainda pela falta de aplicação de padrões de projeto em seus pontos estruturais, propondo então a refatoração, documentações da etapa de análise e levantamento de requisistos, aplicando técnicas de POO e padrões de projeto, permitindo assim a escalabilidade do sistema para desenvolvimentos e manutenções futuras.

1.4 ESTRUTURA DO TRABALHO

O trabalho desenvolvido está apresentado da seguinte maneira:

Capítulo 2 – Referencial Teórico: Econtra-se todo o referencial teórico do trabalho, está dividido em seções que compreendem os conteúdos de revisão sistemática, análise, levantamento de requisitos, Programação Orientada a Objetos, *Unified Modeling Language* (*UML*), Padrões de Projeto.

Capítulo 3 – Material e Metodo: Ferramentas utilizadas no desenvolvimento do trabalho.

Capítulo 4 – Desenvolvimento: Apresenta o desenvolvimento do trabalho

Capítulo 5 – Considerações Finais: Resultados obtidos e cotinuação do trabalho.

2 RERENCIAL TEÓRICO

Neste capítulo é apresentado o referencial teórico deste trabalho, responsável pela fundamentação dos assuntos abordados e tecnologias utilizadas durante o desenvolvimento deste trabalho.

2.1 REVISÃO SISTEMÁTICA DA LITERATURA

A revisão sistemática surgiu da necessidade de aprimorar o processo de levantamento de estudos científicos que sejam relevantes a um determinado assunto, segundo Rf e Mc (2007) “As revisões sistemáticas são desenhadas para ser metódicas, explícitas e passíveis de reprodução”. Sendo assim nota-se que a revisão sistemática não só permite aprimorar o processo de levantamento de conteúdos, como também mostrar um caminho à ser seguido para futuras pesquisas.

Por ser um método científico, a revisão sistemática requer delimitações claras e objetivas sobre determinados aspectos, para que os estudos escolhidos sejam diretamente relacionados ao tema da revisão sistemática, pode-se considerar estes aspectos da revisão sistemática em forma de fases e etapas, como apresentado na Figura 1, apresentando o modelo à serem seguidos para o desenvolvimento de uma revisão sistemática.

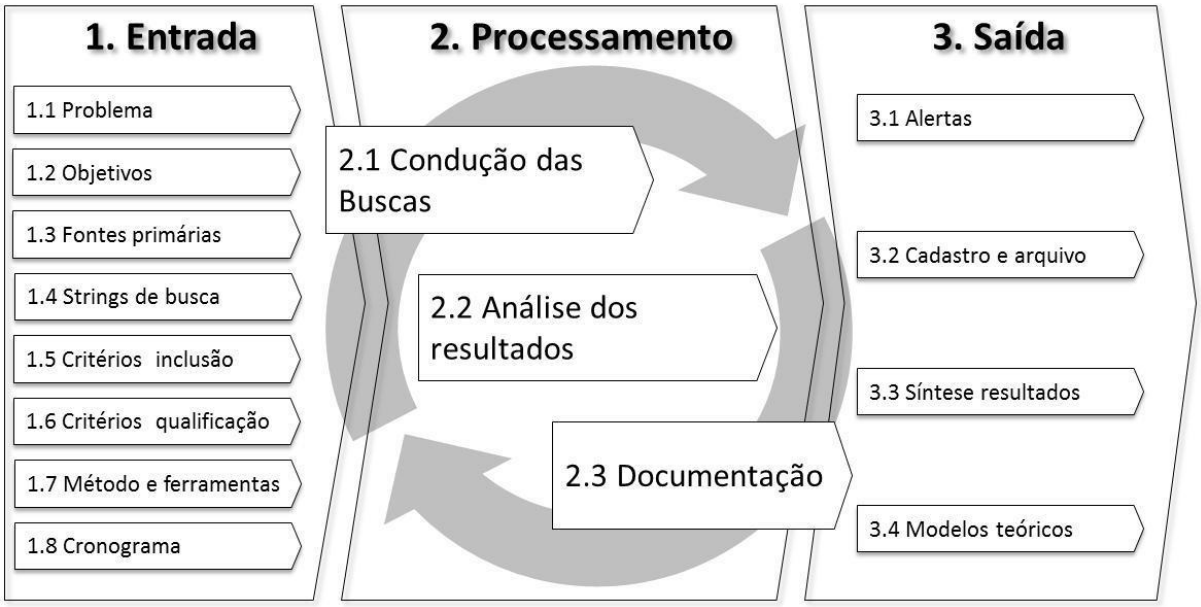

Figura 1- Modelo de condução de uma revisão sistemática
Fonte: CONFORTO (2011).

Na fase de entrada, existem as etapas para a definição do problema, os objetivos que a revisão sistemática pretende esclarecer, a definição da fonte dos estudos, por meio deste, serão encontrados os estudos que podem ser considerados relevantes para a revisão sistemática, a definição da *string* de busca, que será composta por palavras chave juntamente com operadores lógicos, a definição dos critérios de exclusão e inclusão, separados em etapas distintas, estes critérios servem como referência sobre o escopo da revisão sistemática, delimitando assuntos, resultados. Na etapa de métodos são definidos aqueles em que serão realizadas as buscas pelos estudos, armazenamento e filtros de busca. Na última etapa, existe o cronograma, o qual são definidos os prazos, aquisição de *softwares*, parcerias e equipamentos utilizados na elaboração da revisão sistemática.

A segunda fase da revisão sistemática é descrita como processamento que irão realizar o refinamento dos estudos. A etapa de busca é realizada com base nas informações da fase de entrada, de modo a serem efetivamente adquiridos os estudos. Na etapa de análise concentra-se o maior esforço, é realizada manualmente, de modo que os pesquisadores irão analisar cada estudo encontrado, classificá-lo de acordo com os critérios de inclusão ou exclusão e concluir o processamento realizando a última etapa que é a documentação dos estudos

relevantes. Esta fase pode ser cíclica, repetindo-se quantas vezes os pesquisadores julguem necessária para a seleção de estudos efetivamente relevantes.

Na terceira e última fase, a saída é composta pelos alertas, cadastro de arquivos, síntese dos resultados e modelos teóricos. Os artigos identificados como relevantes serão marcados para que tenham prioridade sobre os outros estudos (alertas). O cadastro e arquivo dos artigos que se adequaram aos critérios na fase de processamento, serão efetivamente armazenados para as pesquisas. A síntese dos resultados demarca a confecção de uma síntese dos estudos, contendo os principais autores, artigos, estudos de caso e etc.

A condução do processo de revisão bibliográfica sistemática é algo interativo, de modo que os envolvidos sempre estarão interagindo direta e indiretamente, deste modo a proposta de desenvolver uma ferramenta colaborativa para a elaboração deste processo vem de encontro com um dos requisitos fundamentais da revisão ibliográfica sistemática, ampliando os horizontes de possibilidades para a interação e condução do processo.

Ao propor o desenvolvimento de uma ferramenta que disponibilize a colaboratividade como um de seus serviços principais, amplia-se a gama de possibilidade para a interação e elaboração dos processos de revisão bibliográfica sistemática, oferecendo a possibilidade de unificar pesquisadores de diferentes locais sob um objetivo comum.

2.2 ANÁLISE ORIENTADA A OBJETOS

A análise enfatiza a investigação do problema. O objetivo é levar o analista a investigar e a descobrir. A etapa de análise pode se considerar como a porta de entrada para o desenvolvimento de um *software*, é nesta etapa que se busca a representação do problema, se desenvolvida com qualidade gera um enunciado claro para o problema que se busca resolver e como consequência, o *software* desenvolvido corresponde às expectativas, análises mal elaboradas podem suprir a demana do problema, mas dificilmente atingirá um nível de qualidade satisfatório. O autor ainda afirma “No entanto a primeira fase da análise ocorre uma única vez, e seu objetivo é produzir uma compreensão ampla, mas pouco profunda sobre o

sistema.”, em resumo, fornece ao analista uma visão geral sobre o problema proposto e à partir disso serão desenvolvidos os ciclos de um modo detalhado, implementação e fase de testes (WAZLAWICK, 2004).

2.2.1 Levantamento de requisitos

O levantamento de requisitos, poder ser dividido em etapas distintas, a etapa de análise de requisito e o levantamento dos requisitos propriamente dito.

A análise de requisitos está associada ao processo de descobrir quais são as operações que o sistema deve realizar e quais são as restrições que existem sobre elas (WAZLAWICK, 2004). Nesta etapa são identificadas as necessidades relacionadas ao sistema, descrevendo as necessidades de cada processo encontrado, os elementos externos envolvidos direta e indiretamente com o sistema, fornecendo uma visão geral dos requisistos à serem posteriormente identificados e elaborados.

Na etapa de levantamentos de requistos são descritos de maneira detalhada os requisitos encontrados na etapa de análise dos requisitos ou acrescentados novos requisitos, elaborando um documento com os requisitos devidamente identificados e descritos, com suas respectivas dependências.

2.2.1.1 Requisitos

Requisitos são os elementos identificados na etapa de análise, os quais representam os elementos notórios de um sistema, descrevendo funcionalidades, condições específicas ou ainda informações adicionais. Geralmente os requisitos funcionais identificados são descritos em uma tabela, identificados por um código de representação, sua descrição, requisitos não-funcionais relacionados. Os requisitos suplementares são representados por meio de uma única tabela com suas respectivas descrições.

- Requisitos Funcionais: compreendem as funcionalidades de um sistemas, geralmente identificados como atividades que o sistema tem de cumprir, especificando atividades de grande importância para o sistema, são representados por um código de representação (usualmente começa-se o código com a letra F seguido por um número em sequência crescente), seu nome, descrição e atributos como se o requisito é oculto (se estará perceptível aos clientes que interagirem com o *software*);
- Requisitos não-funcionais: são requisitos que sempre estarão relacionados à um requisito funcional e descrevem como o sistema deve cumprir as atividades propostas de cada requisito funcional, são representados por linhas inseridas na tabela de requisitos funcionais, são representados por um código (usualmente inica-se com as letras NF, seguido pelo número do requisito funcional ao qual representa e um número sequencial crescente, separados por um ponto), sua descrição, e atributos como se o requisito é desejável (se é obrigatório o cumprimento desde requisito) e se o requisito é permanente (se o requisito proposto sempre será tratado desta maneira).

O Quadro 1 apresenta o exemplo de um requisito funcional e seus requisitos não-funcionais.

Quadro 1 - Quadro de requisitos funcionais e não-funcionais de cadastro de usuários

F1 Registrar Usuários		**Oculto ( )**		
Descrição: O sistema deve atender solicitações de cadastros de novos usuários				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF1.1 Validar Cadastro	O cadastro deve conter número de documentos de registros como CPF	Segurança	**(X)**	**(X)**
NF1.2	O identificado de *login* do usuário será o seu endereço de email	Performance	**(X)**	**(X)**
NF1.3	A senha fornecida pelo usuário deverá conter no mínimo 8 caracteres e ser criptografada	Segurança	**(X)**	**(X)**
NF1.4	O usuário deverá estar relacionado a uma instituição de ensino	Implementação	**(X)**	**(X)**

Fonte: Autoria Própria.

2.2.2 Unified Modeling Language - UML

"O *UML*(*Unified Modeling Language*) é uma linguagem para especificação, construção, visualização e documentação de artefactos de um sistema de *software*."(SILVA; VIDEIRA, 2001).

A *UML* foi organizada e compilada por Ivar Jacobson, Grady Booch e Jim Rumbaugh em 1994 que unificaram os conceitos de desenvolvimento e diagramação de modelos orientados a objetos, fundando assim o padrão seguido até a atualidade em desenvolvimento e planejamento de softwares orientados a objetos, conforme mostra a Figura 3 a linha do tempo sobre o desenvolvimento e unificação da *UML*.

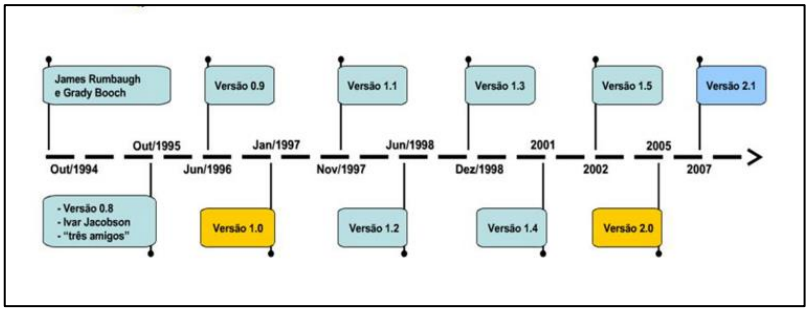

Figura 2 - Linha do Tempo do desenvolvimento da UML
Fonte: Goulart (2007).

Medeiros (2004) apresenta o conceito central da UML como "...a UML é uma forma de comunicar uma idéia...", desta maneira pode-se compreender o ponto central de desenvolvimento da UML, que fornece uma padronização para o desenvolvimento de softwares orientados a objetos, os quais, até o momento de sua criação haviam apenas métodos desenvolvidos de formas divergentes, o que dificultava e muito o desenvolvimento e manutenção do processo de elaboração e desenvolvimento de sistemas, pois havia a necessidade de possuir o conhecimento de diferentes modos de construção de diagramas e como realizar alterações nos mesmos. Atualmente a *UML* é mantinda pela *OMG* (*Object Management Group*) e encontra-se na versão 2.5.

Utilizando a *UML* pode-se elaborar projetos de software sem escrever códigos propriamente ditos, esta modelagem permite desenvolver soluções através de conceitos de orientação de objetos, documentação e diagramas que nos permita contemplar a solução proposta antes que a codificação seja desenvolvida, garantindo assim que problemas relacionados à falta de planejamento e análise compliquem o desenvolvimento do sistema. Para que seja possível representar a soluções encontradas a *UML* oferece diversos artefatos, como por exemplos diferentes tipos de diagramas.

Segundo Silva e Videira(2001) “Os diagramas são conceitos que traduzem a possibilidade de agrupar elementos básicos e suas relações de uma forma lógica ou de uma foram estrutural.”. Diagramas são os elementos oferecidos para representar as soluções desenvolvidas utilizando a *UML*, esta linguagem oferecem diversos diagramas em sua versão atual (versão 2.5, segundo *OMG* (2015)), de modo que estão divididos em dois grupos, diagramas estruturais e comportamentais como apresentado na Figura 4.

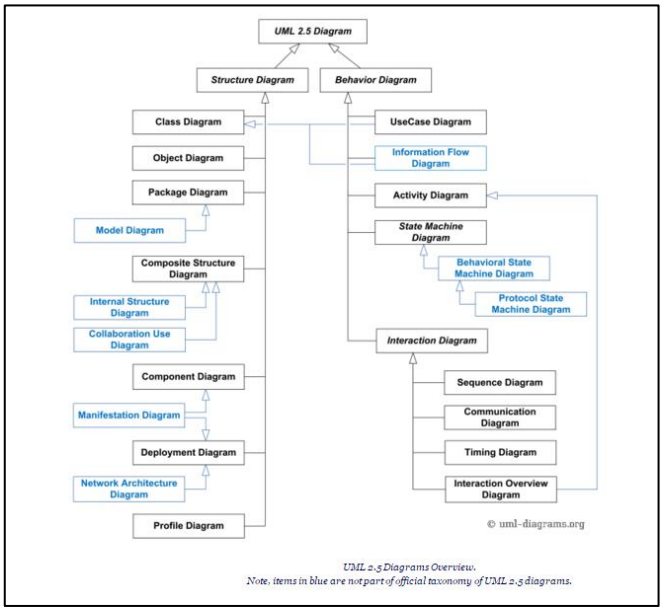

Figura 3 - Estrutura de diagramas UML 2.5
Fonte: UML-DIAGRAMS (2015).

Os diagramas estruturais são responsáveis por apresentar a estrutura básica de uma solução ou projeto, permitindo assim contemplar a orgnanização e o

relacionamento entre os elementos desenvolvidos, os diagramas estruturais estão organizados como:

- Diagrama de Classes;
- Diagrama de Pacotes;
- Diagrama de Componentes;
- Diagrama de Objetos;
- Diagrama de Estrutura Composta;
- Diagrama de Implantação;
- Diagrama de Perfis.

Os diagramas comportamentais representam os fluxos, interações e comportamentos durante um determinado tempo ou evento na solução projetada, facilitando assim a compreensão dos elementos desenvolvidos e estão classficados como:

- Diagrama de Casos de Uso;
- Diagrama de Interação;
- Diagrama de Sequência;
- Diagrama de Comunicação;
- Diagrama de Temporização;
- Diagrama de Interação Geral;
- Digrama de Estados;
- Diagrama de Atividades.

A *UML* é uma linguagem abrangente, que oferece diversas soluções, como o próprio nome implica, é uma linguagem e não um método de desenvolvimento propriamente dito e não implica que seja necessário desenvolver todas as soluções oferecidas para que se obtenha um bom aproveitamento que a *UML* oferece, para cada solução à ser desenvolvida há um diagrama que pode melhor representar à solução buscada.

2.2.2.1 Diagrama de Classes

Este diagrama estutural representa a organização e abstração da classes de uma solução, segundo Kon (2005) que define uma das atribuições de um diagrama de classes como “Modelar as colaborações/interações (socidades de elementos que trabalham em conjunto oferecendo algum comportamento cooperativo)”. Apresentando então um modelo base com diversos elemento como classes, classes abstratas, interfaces, atributos, pacotes, métodos e associações entre estas classes.

- Classes: podem ser representadas de duas maneiras, as classes concretas, que representam por meio de um nome, que irá identificar esta classe, um conjunto de atributos que representam as características de uma determinada classe e seus métodos que pode-se considerar os diversos comportamentos de uma classe e seus modificadores de acessos, estes que definem sua visibilidade em relação à outros objetos que possam se relacionar com estas classes, os elementos de visibilidade são diferenciados como *public*, *private*, *protected*, como apresentado no Quadro 2. Na Figura 5 é representada no diagrama de classes uma classe concreta (*Object*), com seus atributos e métodos, os atributos por sua vez são apresetados por um nome que irá identificá-los seguidos pelo símbolo “:“ que irá determinar qual o tipo deste atributo, podendo ser um tipo simpes (números, textos, caracteres e etc.) ou tipos complexos como exemplo outros objetos complexos;
- Classes Abstratas: estas por outro lado, não são classes concretas, possuem a função de representar um conjunto de classes concretas que possuam características em comum, evitando deste modo que haja duplicidade de código. As classes abstratas transmitem seus elementos por meio de relações de generalização, replicando assim suas características para as classes que as herdam, como apresentado na Figura 5;
- Interfaces : assim como as classes abstratas, não podem ser instanciadas como objetos concretos e representam parcialmente os comportamentos comuns à outras classes, diferindo das classes abstratas, não possuem atributos, seguindo as orientações das boas práticas de programação, havendo apenas os métodos declarados, sem nenhuma implementação concreta, fazendo assim com que as interfaces se assemelhem à um

contrato, dizendo à classe que possui este contrato que existem métodos necessários que a mesma deve realizar suas respectivas implementações, determinando assim que as classes que implementem estas interfaces, construam os métodos de acordo com suas respectivas necessidades, na Figura 5 é apresentada uma interface com seus respectivos métodos declarados no diagrama de classes.

- Pacotes: um diagrama de classes sempre será relacionado à um pacote que representará o elemento raiz por assim dizer, apresentando sempre uma estrutura baseada em um arquivo geral, que irá conter todos os elementos relacionados, um diagrama de classes ainda assim poderá possuir outros pacotes inseridos para representar a utilização de um subconjunto de elementos externo;
- Associações: são relacionamentos que apresentam a dependência subjetiva entre duas classes, aprensentando uma necessidade de relacionamento entre as classes em tempo de execução, mas não exigindo dependências diretas entre ambas;
    a) Agregação: são relações que apresentam uma dependência moderada entre as classes, de modo que uma determinada classe possui os objetos agregados de uma ou mais classes de outro tipo, porém, não dependende existencialmente destas outras classes agregadas;
    b) Composição: é uma relação que indica a dependência de existência entre duas classes, identificando assim a necessidade de que uma classe exista para que a outra também exista;
    c) Generalização: representa como o próprio nome propõe, uma generalização de características, que são herdadas de uma classe concreta (que aplica o princípio de herança, uma das caracterísica da POO) permitindo assim existir uma hirerarquia de classes que generalizam outras classes, porém este fluxo de generalização será sempre unidirecional, nunca cíclico.

Quadro 2 - Atributos de visibilidade dos elementos

Símbolo	Descrição	Significado
-	São visíveis apenas para a própria classe e seus métodos	*Private*
+	São visíveis a quaisquer outras classes ou pacotes	*Public*
#	São visíveis apenas para todas as classes do mesmo pacote(*Package*)	*Protectec*

Fonte: Autoria Própria.

A figura à seguir apresenta uma estrutura básica de um diagrama de classes com um exemplo de utilização de classes concretas, abstratas e interfaces, apresentando seus atributos, métodos e relacionamentos.

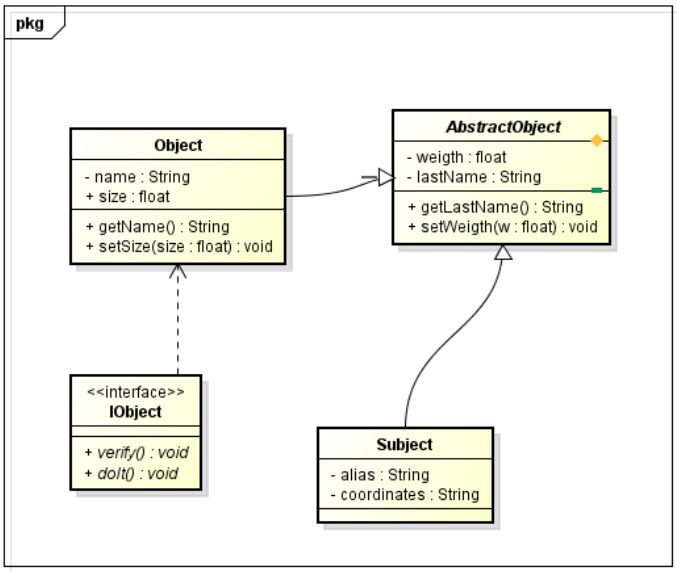

Figura 4 - Diagrama de classes para apresentar os elementos UML
Fonte: Autoria Própria.

2.2.2.2 Diagrama de Sequência

Guedes (2007) descreve este diagrama de comportamento como “O diagrama de sequência preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo.”, em reusmo, este diagrama tem seu foco no fluxo de interação entre os objetos durante um

determinado processo, permitindo assim a visualização contextual sobre os objetos, suas interações, ciclos de vida e trocas de mensagens, usualmente cada diagrama de squência é baseado em um caso de uso identificado no sistema.

- *Lifeline*: cada objeto de interação é representado por uma linha do tempo, o qual pode ser descrito como um ator (que interagem com o sistema) ou como uma interface de acesso ao sistema, controladores de acesso às classes, classes concretas e etc, como apresentado na Figura 6 é apresentado o exemplo de um diagrama de sequência baseado em um caso de uso que requer o registro dos últimos acessos dos usuários de um determinado sistema, de modo a apresentar os elementos envolvidos e seus ciclos de vida;
- *Messages*: as mensagens representam os fluxos de chamadas entre os objetos, podem representar chamadas para a criação ou deleção de objetos, chamadas de métodos para o próprio objeto ou objetos externos ou ainda mensagens de retorno, um ponto importante sobre as mensagens de chamada, são as mensagens sícronas e assíncronas. As mensagens síncronas definem que o objeto que chamou o método, tem de aguardar a sua conclusão para ser liberado a enviar novas mensagens, as mensagens assíncronas permitem uma execução simultânea de várias ativdades independentes que realizam diferentes processamentos e geram resultados independentes para o objeto.

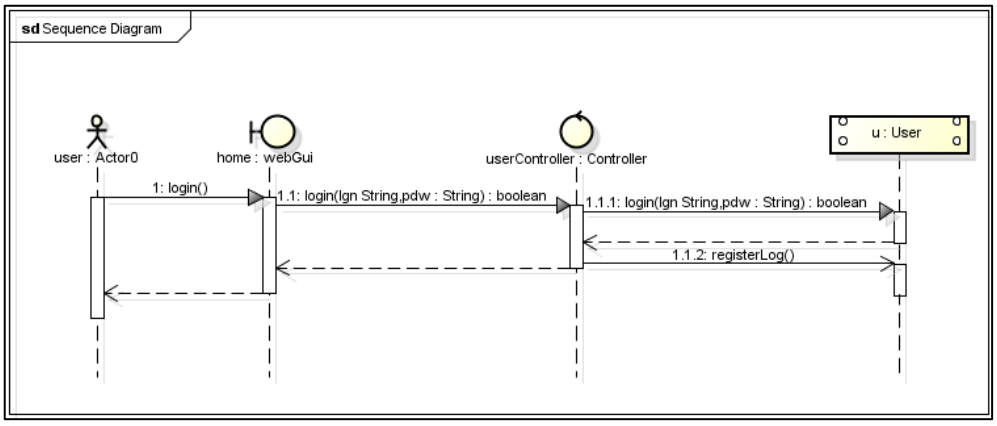

Figura 5 - Diagrama de sequência de login e registro em log
Fonte: Autoria Própria.

2.2.2.3 Diagrama de Casos de Uso

Segundo Guedes(2007) “O Diagrama de Casos de Uso apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma idéia geral de como o sistema irá se comportar.”, tornando este um dos diagramas mais importantes para o desenvolvimento da etapa de análise e levantamento de requisitos, pois permite transimitir de forma prática todas as necessidades identificadas em um sistema, permitindo assim contemplar o seu escopo e seu nível de crescimento.

O diagrama de casos de uso representa os requisitos funcionais identificados na etapa de análise, por meio de interações entre os atores e os casos de usos propriamente ditos.

- Atores: são elementos que representam os usuários externos (pessoas) que interagem diretamente com as funcionalidades propostas pelo sistema, por meio dos casos de uso, estes podem elaborar requisições à um caso de uso e receber uma resposta deste processamento ou simplesmente realizar requisições;
- Casos de Uso: são os elementos que representam os requisitos funcionais de um sistema, representados graficamente por elipses, os requisitos compreendem desde pequenas exigências em determinados pontos de um processo do sistema ou o processo como um todo;
- *Includes* e *Extends*: representam as interações entre casos de uso, os *includes* definem uma relação de dependência entre dois casos de uso, que determinam que para o executar o caso de uso *x* o caso de uso *y* tem de ser executado também, por outro lado as interações definidas como *extends* geram uma relação de não-obrigatoriedade entre os casos de uso, definindo que quando o caso de uso *x* for executado, mediante alguma situação determinada o caso de uso ão determinada o caso de uso *y* será executado. Na Figura 8 é apresentado o exemplo de um diagrama de casos de uso, em que um *ator* usuário, solicita o empréstimo de um livro e este empréstimo é realizado por um *ator* funcionário.

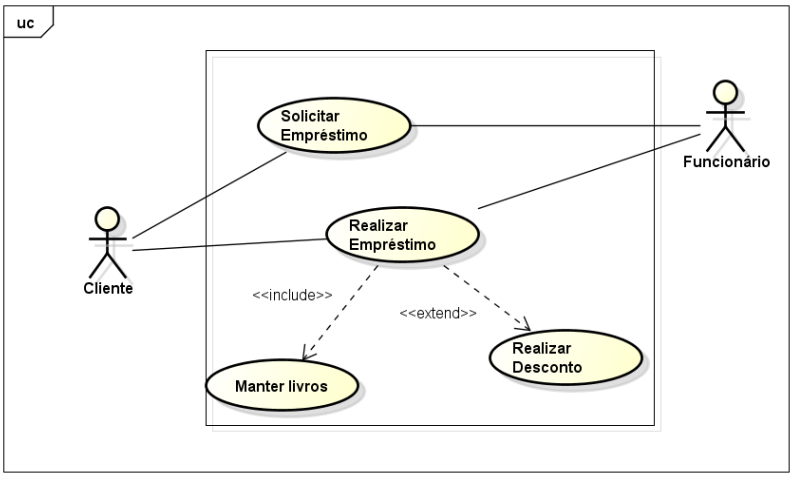

Figura 6 - Diagrama de Casos de Uso de um Sistema de Empréstimos
Fonte: Autoria Própria.

2.2.2.4 Diagrama de Atividades

Silva e Videira (2001) descrevem o diagrama de atividades como,

> *"Um diagrama de atividades é um caso particular de um diagrama de estados, no qual todos ou a maioria dos estados são "estados de atividades" e todas ou a maioria das transições são desencadeades pela conclusão das atividades dos estados anteriores. ".*

Este diagrama representa o fluxo entre as atividades que o sistema oferece, permitindo assim a vizualização das fronteiras de cada área, destinação das atividades que estão atribuídas a cada área determinada.

- Atividades: são representadas por um retângulo e descrevem uma ação ou reposta que o sistema possui;
- Fluxos: são linhas com setas direcionais que indicam o fluxo das chamadas;
- Nós de decisão/União: representados por um losango (diamante) estes nós podem representar condicionais com uma entrada e possíveis saídas ou mais de uma entrada com apenas uma saída;
- Nó de início: representa o início da atividade no diagrama;
- Nó de Finalização da atividade: indica o fim da atividade no diagrama;
- Nó de Fim de Fluxo: representa o fim de determinado fluxo;

- *Forks/Joins*: são representados por uma barra retangular, no caso dos *forks* representam o início de dois fluxos de atividade simultâneos e as *joins* que representam a conclusão simultânea de dois fluxos.

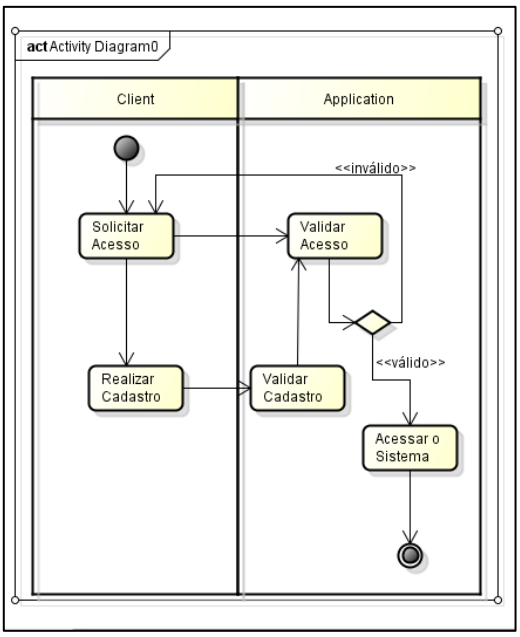

Figura 7 - Diagrama de Atividades de um login e cadastro de um sistema
Fonte: Autoria Própria.

Na Figura 8 é apresentado um exemplo do diagrama de atividades, demonstrando o fluxo de uma atividade de *login*, validação e cadastro em um sistema.

2.2.3 Padrões de Projeto

O conceito de identificação e criação de soluções de maneira padronizada surgiu no ano de 1977 com *Christopher Alexander* ao publicar um catálogo apresentando 250 soluções para problemas comuns na área de engenharia civil e arquitetura. Somente no ano de 1995 a área de *softwares* foi contemplada com o estudo e aplicação de uma padronização de soluções, o livro *Design Patterns: Elements of Reusable Object-Oriented Software* desenvolvido por *Eric Gamma*, *Richard Helm*, *Ralph Johnson* e *Jon Vlissides* mais conhecidos como *Gang of Four*

(*GoF*), tornando-se assim a principal referência em padrões de projeto para desenvolvedores e projetistas.

Guerra (2012) afirma que, “Nossas soluções são expressas em termos de objetos e interfaces em vez de paredes e portas, mas no cerne de ambos os tipos de padrões está a solução para um problema num determinado contexto”. O catálogo identifica 23 padrões de projeto, diferenciando-os em três grupo, padrões de criação, estrutura e comportamento, apresentando cada um, baseando-se em quatro requisitos que todo padrão apresentado deve possuir.

- Nome;
- Objetivo, Intenção ou ainda Motivação;
- Problema;
- Solução;
- Consequências.

Todo padrão deve apresentar um nome que irá identificá-lo, seguido pelo problema que o padrão é proposto a solucionar (neste caso, o objetivo, intenção ou motivação). A aplicabilidade do padrão para identificar as situações em que pode ser empregada a solução frente o problema encontrado, a representação estrutural das classes do padrão apresentando o método de solução do problema, seus participantes (classes, objetos) e a interação entre os elementos que compõem o padrão e suas responsabilidades, consequências de sua aplicação em um projeto, detalhes e aspectos de sua implementação, exemplo de código e em aplicações de diferentes áreas o qual foi aplicado o padrão e quais padrões podem ser utilizados de forma conjunta ou ainda quais são padrões similares e suas diferenças.

Guerra(2012) ainda enfatiza que “Padrões não se refletem em pedaços de código ou componentes que são reutilizados de forma igual em diversas aplicações, eles são um conhecimento que deve estar na cabeça dos desenvolvedores.”, demonstrando assim que os padrões estão muito mais para conceitos de soluções que para um roteiro prático, esta característica conceitual permitiu sua expansão para as mais diferenes áreas. Os padrões de projeto ainda segundo Guerra (2012) “...um padrão não descreve qualquer solução, mas uma solução já tenha sido utilizada com sucesso em mais de um contexto.”. Consolidando asim o o foco conceitual dos padrões, pois muito além da proposta da solução é sempre necessário averiguar o contexto ao qual está tentando se aplicar o padrão, no

próprio catálogo *GoF* existem padrões com propostas e estruturas semelhantes, porém com focos de contexto distintos. Ao aplicar os conceitos de padrões, a habilidade de abstrair estes contextos são os pontos fundamentais para o sucesso da aplicação dos padrões de projeto.

2.2.3.1 Observer

Freeman e Freeman (2007) definem o padrão *observer* como “O padrão observer define a dependência um-para-muitos entre objetos para que quando um objeto mude de estado todos os seus dependentes sejam avisados e atualizados automaticamente.”. Este padrão oferece uma solução para a notificão de objetos quando um objeto principal é modificado, Guerra (2012) ainda complementa “ Além disso, a classe normalmente oferece métodos que permitem a adição e remoção destes observadores em tempo de execução.”, demosntrando assim o controle que a classe principal (observada) tem sobre seus observadores. O padrão *observer* tem como função prover uma ligação leve entre objetos, tornando o código flexível e diminuindo seus impactos para futuras alterações, fazendo com que não exista uma dependência direta da classe observada e as classes observadoras, cada uma não conhece diretamente a classe com a qual de relaciona, apenas a interface *observer* que exerce a dependência, como apesentado na Figura 9. Segundo Freeman e Freeman (2007) “Projetos levemente ligados permitem construir sistemas orientados a Objetos flexíveis que podem lidar com mudanças porque minimizam a interdependência entre os objetos.”.

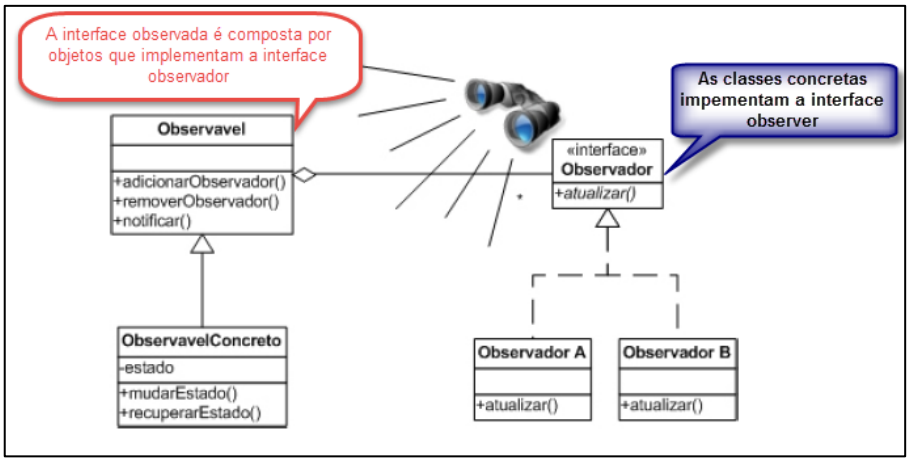

Figura 8 - Estrutura do padrão observer
Fonte: adaptado de Guerra (2012).

2.2.3.2 Template Method

O Padrão *Template Method* define o esqueleto de um algoritmo dentro de um método, transferindo alguns de seus passos para as subclasses (FREEMAN E FREEMAN,2007).

Este padrão oferece uma solução para a necessidade de definir explicitamente um roteiro de passos que devem ser definidos de acordo com cada situação, permitindo assim que a ordem dos passos a serem seguidos seja alterada baseada em cada situação, dando flexibilidade ao sistema pois permite a inclusão de novos casos em que novas ordens de passos serão definidas sem a necessidade de alterar o que já está desenvolvido. Na Figura 10 é apresentado um modelo de classes utilizando o padrão *Template Method* que demonstram as dependências entre a implementação concreta e a classe abstrata que possui os métodos as serem seguidos e como a classe cliente cria um objeto do tipo *template*.

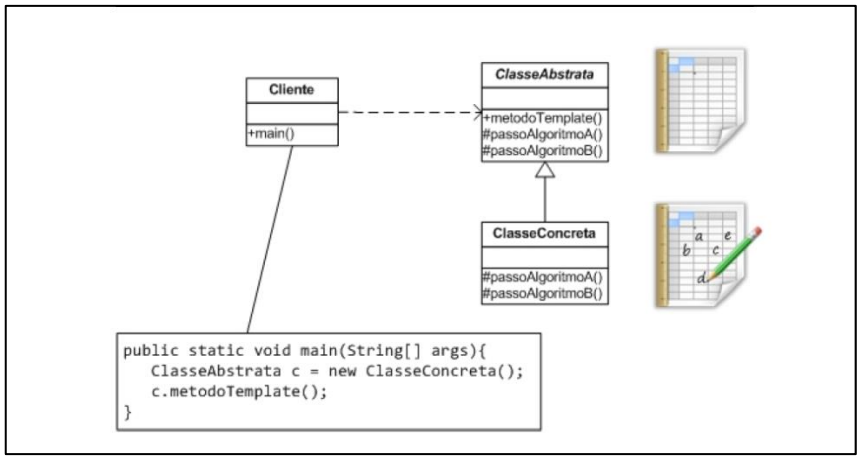

Figura 9 - Diagrama de Classes aplicando o padrão Template Method
Fonte: Guerra(2012).

2.2.3.3 Facade

Guerra(2012) afirma que " A partir deste padrão é possível isolar um conjunto de classes do resto da aplicação deixando a fachada como o único ponto de contato.".

O padrão *facade* permite que existam inúmeras classes dentro de um sistema para relizar um determinado processo, porém o cliente que irá utilizar este processo, irá conhecer somente uma classe de fachada com métodos simples, diminuindo a complexidade de acesso, segundo Freeman e Freeman (2007) descrevem o objetivo do padrão *facade* como "O Padrão Facade fornece uma interface unificada para um conjunto de interfaces em um subsistema", que permitem que uma classe de fachada é quem fique responsável pela interação com as classes de negócio do sistema, fornecendo ao cliente a visão transparente, pois o mesmo conhece somente os métodos relacionados à interface de fachada.

A Figura 11 apresenta o exemplo de um diagrama com o padrão *facade* aplicado, demonstrando como ocorre a interação entre o cliente e o sistema.

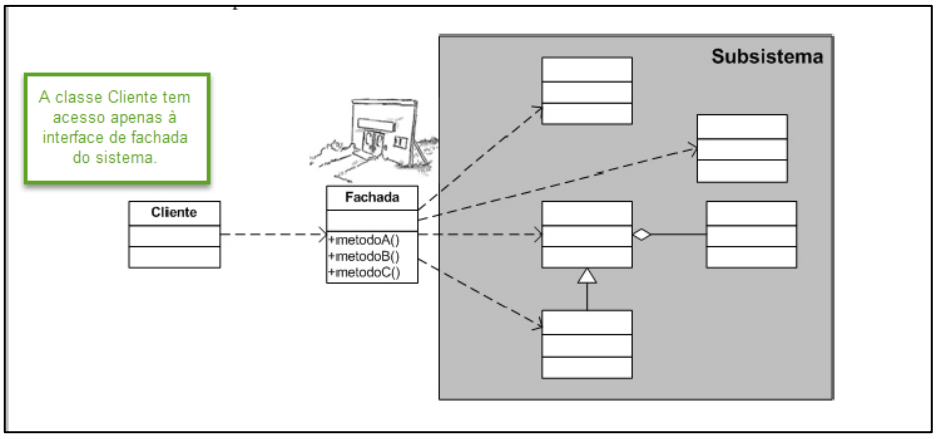

Figura 10 - Diagrama de classes aplicando o padrão Facade
Fonte: Guerra(2012).

2.2.4 Princípios de Projeto Reutilizável Orientado a Objetos

Desenvolvido pelo GoF, estes são princípios baseados nos princípios da POO.

> *Se o programador seguir esses dois não tão complexos princípios, muito provavelmente estará mantendo os princípios básicos da programação orientada a objetos apresentados por Yourdon, Constantine e Meyer e implementando seus padrões de projeto da forma mais adequada possível(Araújo et al., 2013).*

Estes dois princípios citados resumem as boas práticas para o desenvolvimento de um sistema flexível, escalável, com o objetivo de minimizar os impactos para extensões e manutenções futuras. Os dois princípios de projeto reutilizável estão divididos em dois tópicos apresentador a seguir.

- Programe para uma interface, não para uma implementação: programar para interfaces significa realizar as implementações baseadas sempre em tipos abstatos (classes ou interfaces), evitando os tipos concretos (outras classes concretas), quando existem implementações baseadas em tipos concretos diz-se que existe o acoplamento entre classes, quanto maior é este acoplamento, maior é o nível de ligação e consequentemente a complexidade de manutenção das classes, pois gera uma estrutura engessada, com uma cadeia de elementos que dependem diretamente de outros elementos. Este princípio vem para solucionar este problema, oferecendo meios de extender as implementações de tipos generalizados, que representem características

comuns entre as classes, permitindo assim a reusabilidade de código, que é um dos princípios da POO, simplificando a extensão e manutenção do sistema, pois depender de supertipos implica em abstrair os comportamentos comuns, realizar o encapsulamento de suas características e replicar estes elementos para as implementações concretas.

- Prefira Composição de Objeto à Herança de Classe: dar preferência à composição de objetos à herança de classes ajuda a manter cada classe encapsulada e focaliza em uma única tarefa (GAMMA et al., 2000). A composição permite que as classes possuam um foco definido, limitando o seu crescimento ao seu foco, como Gamma(2000) enfatiza sobre a utilização da composição.

> *"Suas classes e hierarquias de classes se manterão pequenas, com menor probabilidade de crescerem até se tornarem monstros intratáveis. Por outro lado, um projeto baseado na composição de objetos terá mais objetos (embora menos classes), e o comportamento do sistema dependerá de seus inter-relacionamentos ao invés de ser definido por uma classe".*

- O desenvolvimento baseado no princípio de composições à heranças permite que um sistema obtenha flexibilidade em termos de comportamento, fazendo com que as classes sempre dependam de tipos abstratos , que representam um conjunto de elementos comuns, reduzindo assim o acoplamento entre classes de modo que o mesmo não tenha o seu comportamento baseado em uma única classe concreta, inviabilizando o crescimento do *software*. Em alguns casos não será possível a utilização de composição e utilizar herança não é uma prática incorreta, porém, assim como quaisquer outros artefatos, é necessário critério e estudo sobre como utilizá-la da forma correta.

2.2.5 Programação orientada a Objetos (POO)

A linguagem de POO foi desenvolvida na década de 60, derivada de uma outra linguagem conhecida como *Simula* que era empregrada para desenvolver simulações, este conceito de simulação do mundo real em abiente computacional passaria despercebido até o desenvolvimento da linguagem *SIMULA-68*, sendo a primeira linguagem que implementou efetivamente os conceitos de POO.

Somente após o desenvolvimento da linguagem *Smalltalk* que a POO recebeu maior visibilidade e por consequência sua popularização, segundo Leite e Júnior (2015) ”Fundamentalmente o que se deseja com esta metodologia são basicamente duas características: reutilização de código e modularidade de escrita”, esta primeira característica fica evidente quando se utiliza os artefatos (classes) que representam coportamentos comuns à outros artefatos, oferecendo o crescimento do modelo lógico sem a perda de performance pela sobrecarga de informações redundantes. A POO ainda segundo Leite e Júnior (2015) “ Formalmente, para que seja considerada uma linguagem *OO*, esta precisa impementar quatro conceitos básicos: abstração, encapsulamento, herança e polimorfismo.”, estes conceitos são as razões de a POO ser algo tão robusto e consciso e estão dividios como:

- Abstração: é considerada como a habilidade de modelar características do mundo ral do problema que o programa estejat tentando resolver. (LEITE; RAHAL JÚNIOR, 2015). Pode ser considerada a percepção para transcrever as características reais da solução que está sendo buscada pelo desenvolvedor, modulando conjuntos de informações que possam representar os objetos reais do levantamento ou problema encontrado, apresentando assim uma solução em ambiente computacional para um problema do mundo real;
- Encapsulamento: é a base de toda a abordagem da Programação Orientada a Objetos; isto porque contribui fundamentalmente para diminuir os malefícios causasos pela interferência externa sobre os dados (LEITE; RAHAL JÚNIOR, 2015). É a garantia que a linguagem oferece meios de reduzir os riscos causados pela interferência de objetos externos sobre os dados de um objeto, este princípio visa definir quais elementos de um objeto (atributos ou métodos) estarão disponíveis para acesso de outros objetos (em diferentes níveis) ou se estarão disponívei apenas para o próprio objeto;
- Herança: é um mecanismo que, se for bem empregado, permite altos graus de reutilizaçãp de código. (LEITE, RAHAL JÚNIOR, 2015). Este princípio permite a ligação entre duas classes, uma delas que é a classe pai que contém determinados atributos e métodos e uma classe filho que por meio da herança, adquire o conhecimento sobre todos os atributos e métodos da classe pai, este conceito se bem aplicado incrementa a reutilização de código

e permite um ganho no desenvolvimento justamente pela reutilização de códigos e replicação de mudanças nas classes pai;

- Polimorfismo: é definido como sendo um códgo que possui “vários comportamentos” ou que produz “vários comportamentos”, em outras palavras, é um código que pode ser aplicado à várias classes de objeto. (LEITE, RAHAL JÚNIOR, 2015). O polimorfismo permite a utilização de métodos que podem ser utilizados em diferentes tipos de objetos, sem que haja problemas frente à diferentes tipos de argumentos de entrada, possuindo diferentes comportamentos em relação aos parâmetros oferecidos durante a sua chamada, este princípio permite a utilização simplificada de métodos, sem que seja necessário criar diferentes métodos para diferentes tipos de entrada, mando assim uma aparência de transparência.

2.2.6 Java

Mendes(2009) descreve a linguagem java como “A linguagem *Java* foi criada seguindo o paradigma da orientação a objetos e, por isso, traz de forma nativa a possibilidade de o programador usar os conceitos de herança, polimorfismo e encapsulamento.”, esta linguagem que surgiu no início da década de 90 como parte de um projeto da empresa *SunMicrosystems* acabou ganhando forma e foi apresentada como uma linguagem de POO no ano de 1995 e com ela a propagação em massa dos conceitos de POO que desde a sua criação na década de 70 ainda não havia credibilidade.

Java é uma linguagem de alto nível, orientada a objetos e multiplataforma, pois não é uma linguagem de programação compilada diretamente na máquina, graças a *Java Virtual Machine* (*JVM*) que é a responsável por interpretar o código e gerar os arquivos para compilação, a linguagem também possui interfaces de aplicação programadas (*Application Programming Interface ou API*) que basicamente são várias bibliotecas com métodos e funções para as mais diversas aplicações, sendo assim, estas características tornam o código independente do *hardware* ou do sistema operacional que está sendo executado, desde que possua a

JVM instalada, a *Figura* 2 apresenta o processo de compilação e execução de um arquivo em linguagem *Java*.

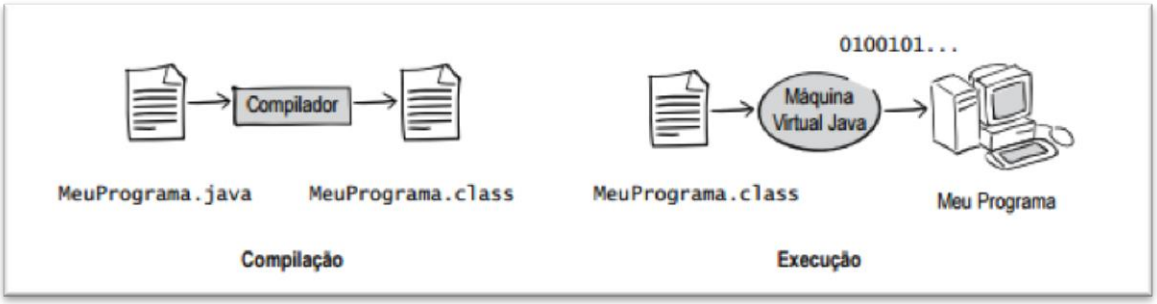

Figura 11 - Compilação e execução de um código Java
Fonte: Mendes (2009).

3 MATERIAIS E MÉTODOS

Como a elaboração visa reformular e readequar o projeto desenvolvido por Ferreira (2014), houve um uso de algumas ferramentas para a criação de diagramas e implementação de código para compreender a aplicação dos conceitos e padrões inseridos no projeto e uma ferramenta para gerar um protótipo de *layout* da ferramenta.

3.1 VERSIONAMENTO

O controle de versão é um meio utilizado para permitir o desenvolvimento conciso de um *software* permitindo que este código receba correções e incrementos de forma segura e prática, para este fim será utilizada a ferramenta *GitHub*, pois é uma ferramenta *OpenSource* de fácil utilização e controle, sendo utilizada como um *plug-in* da ferramenta *netbeans* que oferece suporte ao controle de versionamento.

3.2 ASTAH PROFESSIONAL

Ferramenta *CASE*(*Computer-Aided Software Engineering*) que oferece soluções para a elaboração das etapas de análises orientada a objetos (diagramas), análise estruturada (Modelo Entidade Relacionamento – MER) e ferramentas para otimizar o desenvolvimento, permitindo a geração de códigos em diversas linguagens à partir de diagramas desenvolvidos, geração de engenharia reversa (gerar modelos conceituais à partir de códigos implementados). Foi utilizada a versão *professional* utilizando a licença para estudantes (liberada mediante comprovação de registro acadêmico).

3.3 NETBEANS

É uma ferramenta de ambiente de desenvolvimento integrado (*IDE)* baseada na linguagem *Java*, para desenvolvimento de várias linguagens de programação e marcação (*Java*, *C, C++, Hyper Text Markup Language (HTML), JavaScript, CSS* e *PHP*), integrando uma vasta gama de ferramentas, *templates* e *frameworks* para otimizar o desenvolvimento, é a *IDE* oficial para a versão 8 do *Java* oferecendo suporte oficial para as novas funcionalidades e encontra-se na versão *8.0.2*.

3.4 WIREFRAMESKETCHER

É uma ferramenta disponibilizada para ambiente *desktop* ou *plugin* da *IDE* de desenvolvimento *Eclipse* que oferece a prototipagem da camada de visão para áreas *web* e *mobile*, apresentando quadros com operações de fluxos, fornecendo assim a visão aproximada do produto final. Como é uma ferramenta proprietária, existe um período de teste da ferramental (*trial*) de 15 dias, aos quais foram utilizados para o desenvolvimento dos protótipos apresentados.

3.5 REFATORAÇÃO

O projeto correspondente a ferramenta será revisto e readequado seguindo os conceitos de orientação a objetos, baseando-se na primeira versão elaborada por Ferreira(2014).

Incluindo requisitos relacionados aos módulos de revisão bibliográfica sistemática, definindo novas funcionalidades para o sistema como a integração de *engines* de busca, os processos para a definição dos critérios e palavras chave como processos chave, reforçando deste modo a colaboratividade da ferramenta e a elaboração de um novo módulo social (o qual não havia sido implementado), para que a ferramenta adicione mais funcionalidades à colaboratividade estabelecida

como proposta do projeto, compreendendo assim a análise e aplicação dos conceitos de POO, baseando-se assim em etapas distintas para a elaboração do projeto.

4 RESULTADOS E DISCUSSÕES

São apresentandos os conteúdos desenvolvidos por Ferreira(2014) e as discussões com os resultados obtidos na reestruturação da análise da ferramenta de revisão bibliográfica sistemática.

Desenvolvida por Ferreira(2014) a ferramenta proposta estava com as etapas de levantamento de requisitos e análises do módulo do processo de revisão sistemática concluídas, estando assim em andamento na fase de implementação. A proposta deste estudo foi readequar o modelo proposto por Ferreira(2014), incluir novas funcionalidades, permitindo assim o crescimento escalável da ferramenta e sua continuidade futura, realinhando a ferramenta com os conceitos de POO, aplicando conceitos de princípios de projeto e padrões de projeto, ao qual o objetivo é a fundamentação concisa da ferramenta, tornando-a escalável e com um menor custo de manutenção.

4.1 LEVANTAMENTO DE REQUISITOS

Os requisitos levantados por Ferreira(2014) estão apresentados na Figura 12, que identificou apenas alguns pontos importantes relacionados ao sistema, que compreende o funcionamento da ferramenta como um todo.

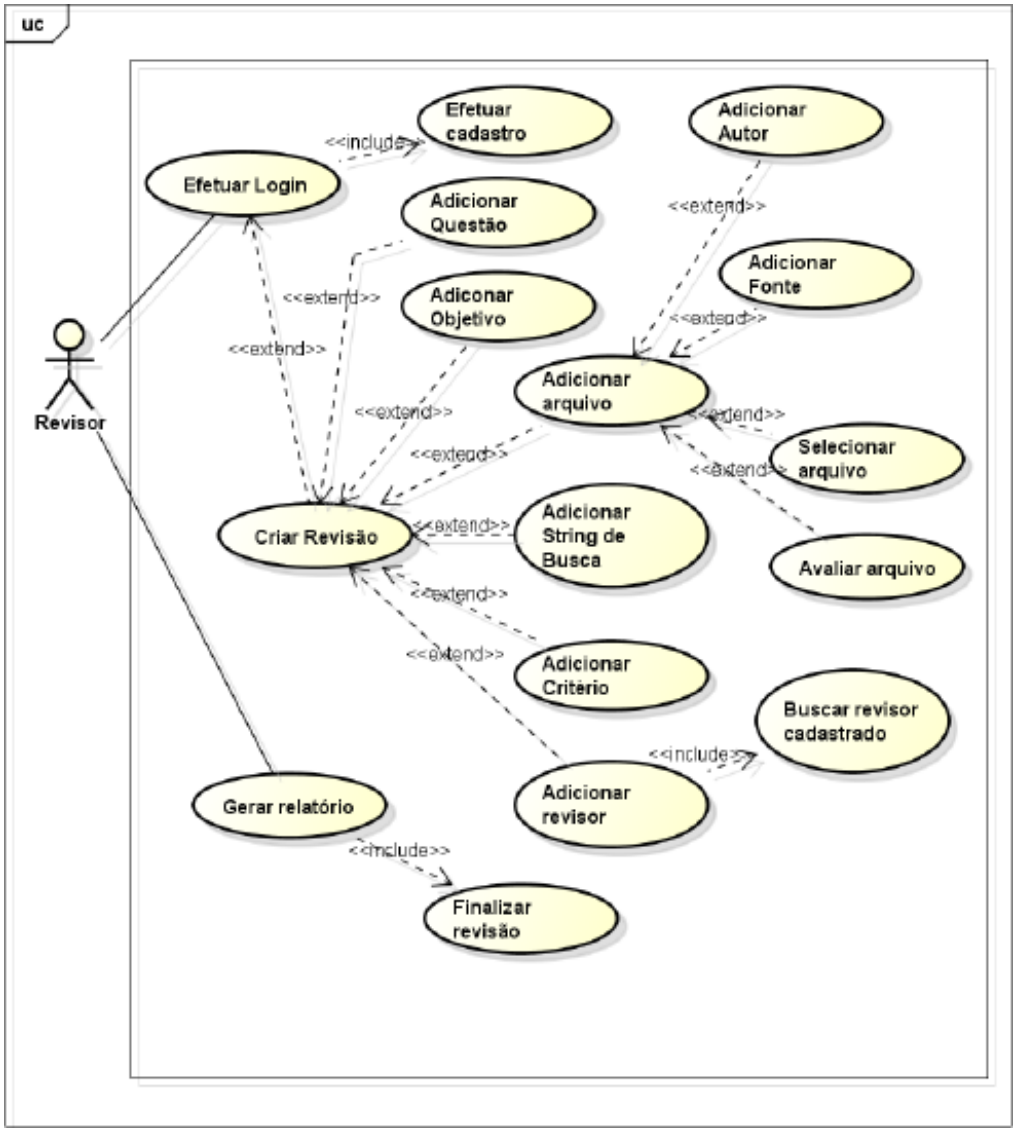

Figura 12 - Casos de Uso elaborados na primeira versão da ferramenta
Fonte: Ferreira(2014).

Na Figura 13 é apresentada a reafotação dos casos de uso do *software*, identificando por meio da análise, com a identificação de novos requisitos, estes descrevem processos de um modo geral, desta forma reduzindo a quantidade de casos de uso identificados em relação ao diagrama elaborado por Ferreira (2014). Também foram incluidos requisitos para a criação do módulo de *Chat*, que permitirá aos usuário a troca de mensagens.

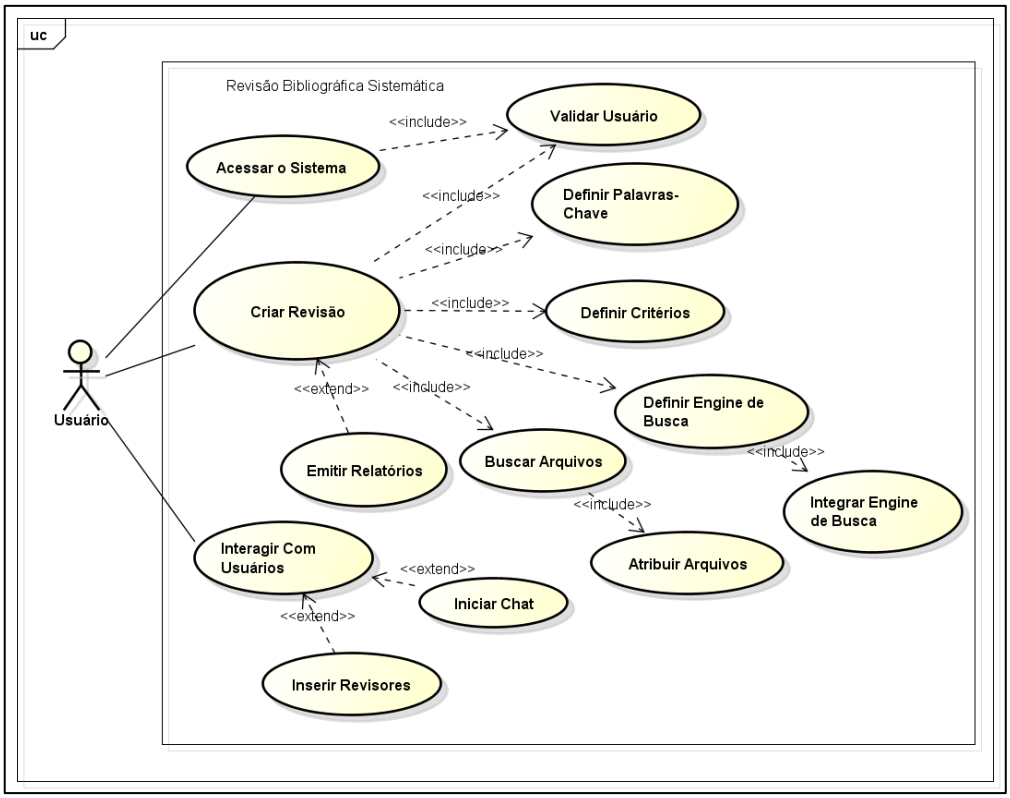

Figura 13 - Casos de Uso Revistos
Fonte: Autoria Própria.

O levantamento dos casos de uso também foi documentado, ao qual será apresentado em forma de tabelas e divididos em quatro áreas principais, os requisitos funcionais e não funcionais relacionados ao processo de revisão sistemática, os requisitos relacionados ao módulo de *chat*, *engine* de busca e os requisitos suplementares, aos quais serão apresentadas as determinações relacionadas ao sistema ao que se referem as tecnologias, ferramentas e meios de armazenamento, os requisitos relacionados à revisão sistemática estão apresentados nos Quadros 3 a 13.

Quadro 3 - Requisito Funcional F1 (Registrar Usuários)

F1 Registrar Usuários		**Oculto ( )**		
Descrição: O sistema deve atender solicitações de cadastros de novos usuários				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF1.1 Validar Cadastro	O cadastro deve conter número de documentos de registros como CPF	Segurança	**(X)**	**(X)**
NF1.2	O identificado de *login* do usuário será o seu endereço de email	Performance	**(X)**	**(X)**
NF1.3	A senha fornecida pelo usuário deverá conter no mínimo 8 caracteres e ser criptografada	Segurança	**(X)**	**(X)**

NF1.4	O usuário deverá estar relacionado a uma instituição de ensino	Implementação	**(X)**	**(X)**

Fonte: Autoria Própria.

Quadro 4 - Requisito Funcional F2 (Validar Usuários)

F2 Validar Usuários		**Oculto ( )**		
Descrição: O sistema exige acesso por autenticação com *login* e senha				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF2.1 Acesso único	O sistema deverá manter o usuário logado enquanto o sistema houver conexão	Performance	**( )**	**( )**
NF2.2 Validação de acesso	Para acessar qualquer conteúdo do sistema o usuário deverá estar logado	Segurança	**(X)**	**(X)**
NF2.3 *Log* de usuários	O sistema deverá armazenar as datas de acessos dos últimos 3 meses de cada usuário	Segurança	**(X)**	**( )**
NF2.4 Usuários ativos	Usuário com mais de 1 ano sem realizar acesso deverão ser considerados inativos	Performance	**(X)**	**( )**

Fonte: Autoria Própria.

Quadro 5 - Requisito Funcional F3 (Criar Revisões)

F3 Criar Revisões		**Oculto ( )**		
Descrição: O sistema permite que usuários criem revisões bibliográficas sistemáticas				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF3.1 Limite de revisões	Cada usuário ser responsável por apenas uma única revisão sistemática e ser revisor em até duas revisões criadas por outros usuários	Implementação	**(X)**	**( )**
NF3.2	O usuário será o revisor responsável da revisão criada	Implementação	**(X)**	**(X)**
NF3.3	Cada revisão deverá possuir no mínimo 2 usuários revisores	Implementação	**(X)**	**(X)**
NF3.4	Cada revisor incluso na revisão deverá receber uma solicitação de aceite de participação	Performance	**(X)**	**(X)**
NF3.5	O sistema deverá permitir criar revisões de revisões anteriores com no mínimo 1 ano de conclusão	Performance	**( )**	**( )**

Fonte: Autoria Própria.

Quadro 6 - Requisito Funcional F6 (Emitir Relatórios)

F4 Emitir Relatórios		**Oculto ( )**		
Descrição: O sistema permite que usuários emitam relatórios à partir de revisões concluídas				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF4.1 Formatos	Os relatórios poderão ser	Implementação	**(X)**	**(X)**

de apresentação	apresentandos em mais de um formato			
NF4.2 Informações do relatório	Cada relatório deverá constar as informações de cada revisor, data de inicio, conclusão, lista de arquivos revisados, arquivos exclusos e gráficos de aproveitamento	Implementação	**(X)**	**( )**

Fonte: Autoria Própria.

Quadro 7 - Requisito Funcional F5 (Listar Revisões)

F5 Listar Revisões	**Oculto ( )**			
Descrição: O sistema fornece um ambiente de acesso a revisões concluídas				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF5.1 Acesso de informações	A lista de revisões deverá fornecer o título da revisão, data de início e revisores	Implementação	**(X)**	**( )**
NF5.2 Meios de acesso	O acesso as informações de cada revisão será por meio de *links*	Segurança	**( )**	**( )**
NF5.3 Filtro de buscas	O sistema deverá oferecer filtros de buscas por títulos de revisões, datas, revisores	Performance	**(X)**	**( )**
NF5.4 Acesso de revisões	Cada revisão acessada para consulta deverá apresentar os documentos selecionados e o relatório referente a revisão	Implementação	**(X)**	**( )**
NF 5.5 Listagem de revisões	Somente revisões concluídas poderão ser pesquisadas	Performance	**(X)**	**( )**

Fonte: Autoria Própria.

Quadro 8 - Requisito Funcional F6 (Incluir Novos Revisores)

F6 Incluir Novos Revisores	**Oculto (X)**			
Descrição: O sistema deve fornecer uma lista de revisores disponíveis para serem incluídos em uma nova revisão.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF6.1 Lista de Revisores	A lista deverá apresentar a lista de revisores à partir de dois tópicos, revisores inseridos na lista de contato de troca de mensagenss e outros revisores ativos no sistema	Implementação	**(X)**	**( )**
NF6.2 Buscar Revisores	A lista deverá conter um campo para filtrar os revisores pelo nome, sendo atualizada a cada caractere inserido	Performance	**(X)**	**( )**
NF6.3 Revisores Ativos	A lista irá apresentar somente revisores que considerados ativos no sistema	Performance	**(X)**	**(X)**

Fonte: Autoria Própria.

Nos Quadros 9 e 10 são apresentados os respectivos requisitos que fundamentam uma das funcionalidades oferecidas pela colaboratividade da ferramenta, tornando ambos os processos pontos chave da condução do processo de revisão sistemática, oferecendo assim atividades que impliquem na colaboração entre os envolvidos.

Quadro 9 - Requisito Funcional F7 (Definir Palavras Chave)

F7 Definir Palavras Chave		**Oculto (X)**		
Descrição: O sistema deve conter uma tela a parte para a elaboração do processo de escolha das palavras chave.				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF7.1 Iniciar escolha das palavras chave	Para inciar o evento de escolha das palavras chave todo os usuários deverão estar conectados e deverão receber um aviso de solicitação de início da etapa, sendo redirecionados para a tela	Implementação	**(X)**	**(X)**
NF7.2 Tela de Palavras Chave	Deverá ser uma tela diferenciada no sistema, de modo que haverá uma sala de troca de mensagens entre os revisores e um outro espaço permitindo assim que cada revisor insira as palavras chave e as submeta, sendo então apresentadas em um quadro visível a todos os revisores	Interface	**(X)**	**( )**
NF7.3 Avaliação de palavras chave	As palavras chave que forem submetidas ao mural, deverão ser avaliadas por todos os outros revisores, para que seja eliminada, uma palavra chave deve ser rejeitada pela maioria dos revisores	Implementação	**(X)**	**( )**
NF7.4 Finalizar etapa de escolha de palavras chave	Caberá ao revisor responsável concluir a etapa de escolha das palavras chave, por meio de um botão na própria tela e confirmando a conclusão da etapa	Segurança	**(X)**	**(X)**

Fonte: Autoria Própria.

Quadro 10 - Requisito Funcional F8 (Definir Critérios)

F8 Definir Critérios		**Oculto (X)**		
Descrição: Ao criar uma revisão o sistema deverá possuir uma etapa relacionada a definição dos critérios de inclusão ou exclusão de um arquivo				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF8.1 Iniciar Etapa para definir critérios	Para iniciar a etapa de escolha dos critérios todos os revisores deverão estar conectados, esta etapa deve ser iniciada pelo revisor responsável e notificada	Implementação	**(X)**	**(X)**

	aos outros revisores			
NF8.2 Tela de Critérios	Em uma tela específica que contém um espaço para troca de mensagens simultâneas entre os revisores, dois murais para as escolhas dos critérios de inclusão e exclusão do estudos, uma área com um campo texto, um botão e uma lista indicando em qual mural a palavra deverá ser submetida	Interface	**(X)**	**( )**
NF8.3 Avaliar Critérios	Para que um critério seja desconsiderado, o mesmo ter de ser avaliado negativamente pela maioria dos revisores	Implementação	**(X)**	**(X)**
NF8.4 Concluir Etapa	Para concluir a etapa, o revisor responsável irá submeter a finalização por meio de um botão na própria tela e realizando uma confirmação para a conclusão	Implementação	**(X)**	**( )**

Fonte: Autoria Própria.

Quadro 11 - Requisito Funcional F9 (Definir Engines de Busca Para Módulo de Busca)

F9 Definir Engines de Busca Para Módulo de Busca	**Oculto (X)**			
Descrição: O sistema deve manter o registro das engines de busca (fontes de dados) onde serão realizadas as pesquisas dos estudos				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF9.1 Cadastrar engines	O cadastro poderá ser realizado apenas por usuários com perfil de administrador da ferramenta e deverão conter informações de credenciais de acesso caso haja necessidade	Segurança	**(X)**	**(X)**
NF9.2 Integrar Engines	Integração da engine de busca com o módulo de *Engines* da ferramenta, de modo que o módulo forneça as palavras-chave e a *engine* de busca retorna os estudos encontrados	Implementação	**(X)**	**( )**
NF9.3 Validar *Engine*	As engines cadastradas na ferramenta devem ser fontes de estudos confiáveis e oferecerem serviços confiáveis de acesso	Performance	**(X)**	**( )**
NF9.4 Engines ativas	Para que uma engine seja listada como ativa no sistema o mesmo deve realizar um teste baseado no tempo de resposta	Performance	**(X)**	**(X)**
NF9.5 Criar busca de dados	O módulo de Busca de Dados da ferramenta submete para a *engine* uma *String* com as palavras-chave definidas e então a engine devolve uma lista com os resultados encontrados	Implementação	**(X)**	**( )**
NF9.6 Armazenar Dados	Os resultados encontrados pela *Engine* de busca deverão ser armazenados pelo sistema	Implementação	**(X)**	**( )**

Fonte: Autoria Própria.

Quadro 12 - Requisito Funcional F10 (Atribuir Estudos)

F10 Atribuir Estudos		**Oculto (X)**		
Descrição: Com a lista total de estudos encontrados o sistema irá distribuir entre todos os revisores da revisão sistemática				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF10.1 Distribuir Estudos	O sistema irá disponibilizar para cada revisor um diretório com os estudos destinados a cada revisor	Armazenamento	**(X)**	**(X)**
NF10.2 Acessar estudos	Cada revisor poderá acessar somente os seus diretórios	Segurança	**(X)**	**(X)**
NF10.3 Listar Estudos	Será criada uma lista para cada revisor com os estudos encontrados	Implementação	**(X)**	**( )**

Fonte: Autoria Própria.

Quadro 13 - Requisito Funcional F11 (Avaliar Estudos)

F11 Avaliar Estudos		**Oculto ( )**		
Descrição: Cada revisor deverá avaliar determinado estudo, oferecendo as justificativas para sua inclusão ou exclusão				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF11.1 Listar Estudos	O sistema irá disponibilizar uma tela específica para os revisores com uma lista de seus respectivos estudos para avaliação	Interface	**(X)**	**(X)**
NF11.2 Selecionar Estudo	O revisor seleciona o estudo da lista e é direcionado para uma tela de avaliação, onde é apresentado um questionário de avaliação, com a inclusão ou exclusão do estudo baseado nos critérios de inclusão e exclusão	Interface	**(X)**	**( )**
NF 11.3 Submeter Avaliação do Estudo	Após preencher o questionário o revisor submete o formulário de avaliação, caso o estudo tenha sido eliminado, será encaminhado a outro revisor	Implementação	**(X)**	**(X)**
NF11.4 Submeter avaliação de segundo nível	O artigo excluido na primeira avaliação é enviado a outro revisor, caso o veredito seja diferente da primeira avaliação, é enviado para o revisor responsável	Implementação	**(X)**	**(X)**
NF11.5 Revisor responsável avalia estudo	O revisor responsável recebe o estudo com as avaliações divergentes, avalia o estudo e submete o veredito final sobre o estudo	Implementação	**(X)**	**(X)**
NF11.6 Excluir Estudo	Quando um estudo é eliminado, dever ser mantido seu link de origem ou Informações como título, ano e autores do estudo	Performance	**(X)**	**(X)**
NF11.7 Armazenar Estudos Aceitos	Os estudos avaliados para inclusão deverão ser armazenados em um repositório relacionado à revisão e de acesso comum	Implementação	**(X)**	**(X)**

Fonte: Autoria Própria.

Os requisitos funcionais e não funcionais correspondentes ao módulo de *chat* estão apresentados nos Quadros 14 e 15.

Quadro 14 - Requisito Funcional F12 (Inscrever Revisores)

F12 Inscrever Revisores		**Oculto ( )**		
Descrição: O sistema deve permitir que revisores adicionem outros revisores em suas listas de contatos				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF12.1 Listar Revisores	Deve ser apresentada uma lista de busca com os revisores considerados ativos no sistema	Implementação	**(X)**	**(X)**
NF12.2 Notificar Revisores	Os revisores inseridos em uma lista de contatos irão aguardar comfirmação de inclusão na lista de contatos	Implementação	**(X)**	**(X)**

Fonte: Autoria Própria.

Quadro 15 - Requisito Funcional F13 (Iniciar Chat)

F13 Iniciar chat		**Oculto ( )**		
Descrição: O módulo do sistema deve permitir a troca de mensagens instantâneas entre os revisores				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF12.1 Identificar Status dos contatos	Na lista de contatos, cada um deverá ser apresentado pelo nome, seguido de um ícone que identifique o seu status	Implementação	**(X)**	**(X)**
NF12.2 Iniciar troca de mensagens	A troca de mensagens só pode ser iniciada a partir de contatos já inseridos e confirmados na lista de cada revisor	Implementação	**(X)**	**(X)**
NF12.3 Enviar mensagem	O revisor poderá abrir a janela de chat com um contato da sua lista e enviar uma mensagem mesmo enquanto o mesmo estiver com status *offline*	Implementação	**(X)**	**(X)**
NF12.4 Notificar mensagens recebidas	Assim que o revisor efetuar o *login* no sistema, deverá ser notificado caso hajam mensagens recebidas	Interface	**(X)**	**( )**
NF12.5 Armazenar troca de mensagens	As mensagens devem ser armazenadas em um arquivo *log* diariamente	Segurança	**(X)**	**( )**

Fonte: Autoria Própria

No Quadro 16 é apresentado o requisito funcional relacionado à engine de *busca*, que define um meio de integração entre *engines* já existentes e a ferramenta, visando o aproveitamento de tecnologias já existentes.

Quadro 16 - Requisito Funcional F13 (Integrar Engine de Busca)

F14 Integrar *Engine* de busca		**Oculto (X)**		
Descrição: O sistema deve possui integração com as *engines* de busca cadastradas no módulo de Busca de Dados				
Requisitos Não-Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF13.1 Verificar Eficiência	As engines disponibilizadas para utilização deverão possui tempos de resposta inferiores a 10 segundos	Performance	**(X)**	**(X)**
NF13.2 Verifica disponibilidade	Uma vez por semana o sistema deverá realizar uma verificação de resposta da *engine* utilizada	Implementação	**(X)**	**( )**
NF 13.3 Validar Credenciais	Caso a *engine* solicita possua credenciais inválidas ou incorretas, deverá ser retirada da lista e notifica aos usuários com perfil de administrador do sistema	Performance	**(X)**	**(X)**

Fonte: Autoria Própria.

O Quadro 17 apresenta os requisitos suplementares, indicando informações relevantes em relação ao desenvolvimento do sistema, identificando necessidades de uso de determinadas tecnologias, armazenamento e controles de acesso.

Quadro 17 - Tabela de Requisitos Suplementares

Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
S1 Interface	As interfaces do sistema serão desenvolvidas em *HTML5* utilizando *framework BootStrap*	Interface	**(X)**	**( )**
S2 Armazenamento	A camada de persistência será desenvolvida utilizando o *framework Hibernate*	Persistência	**(X)**	**( )**
S3 *MVC*	O projeto será dividido em camadas de modelo, visão e controles por meio do *framework Vraptor*	Desenvolvimento	**( )**	**(X)**
S4 Perfil de Acesso	Haverá dois perfis de acesso para o sistema, Usuários (consulta e administração de revisões e interações no módulo social) e Administradores (gerenciamento total da ferramenta)	Segurança	**(X)**	**(X)**

Fonte: Autoria Própia.

O Quadro 18 apresenta as referências cruzadas entre cada caso de uso, isto é, casos de uso que estão diretamente ligado à outros casos de uso, gerando desta maneira uma dependência que deve ser identificada para prever maiores impactos e definir uma estrutura base para a elaboração destes casos de uso.

Quadro 18 - Tabela de Referências Cruzadas de Requisitos

Nome	Atores	Descrição	Referências Cruzadas
Acessar o Sistema	Usuário	Usuário realiza o acesso ou se cadastra caso não o tenha feito	F1,F2
Criar Revisão	Usuário	O usuário irá solicitar a criação de uma revisão sistemática	F3, F5, F6, F10, F11, F13
Interagir Com Usuários	Usuário	O usuário irá interagir socialmente com outros usuários por meio do sistema	F12, F13
Emitir Relatórios	Usuário	O usuário solicita relatórios sobre determinada revisão	F4
Definir Palavras-Chave		Etapa de definição das palavras-chave para relizar buscas	F7
Definir Critérios		Processo de escolha de critérios de inclusão e exclusão de um estudo	F8
Definir *Engine* de busca		Processo para escolha e execução das buscar utilizando *engines*	F9

Fonte: Autoria Própria.

A documentação dos requisitos é um processo tão essencial quanto a própria identificação dos requisitos, pois fornece um nível de detalhamento maior, que visa extinguir más interpretações em relação aos casos de uso identificados, deixando-os claros e diretos e permitindo assim um controle sobre a evolução dos próprios casos de uso e do *software* desenvolvido.

4.2 ANÁLISE

Neste ponto identificou-se o maior problema em relação ao *software* desenvolvido até então, uma estrutura de classes apresentada de forma engessada, impossibilitando a adição de novos elementos sem ampliar exponencialmente o custo de manutenção, como por exmeplo a classe `Arquivo`, caso haja a necessidade de criar novas classes para diferenciar os tipos de arquivos existentes, será necessário replicar a inclusão dos novos tipos em todas as classes que possuem um objeto `Arquivo`, tornando inviável a correção do modelo existente pelo alto custo de tempo demandado para tal. Também não houve a identificação de aplicação dos conceitos de POO, padrões de projeto ou mesmo os conceitos de princípios de projeto.

4.2.1 Modelo Proposto

Na Figura 14 é apresentado o diagrama de classes proposto por Ferreira (2014), analisando a viabilidade de correção do modelo foi decido pela elaboração de um novo modelo que atenda os requisitos de flexibilidade, padronização, boas práticas e extensibilidade, fazendo assim com que o novo modelo proposto atenda aos requisitos e possua um ciclo de vida maior frente às mudanças que podem surgir durante seu desenvolvimento.

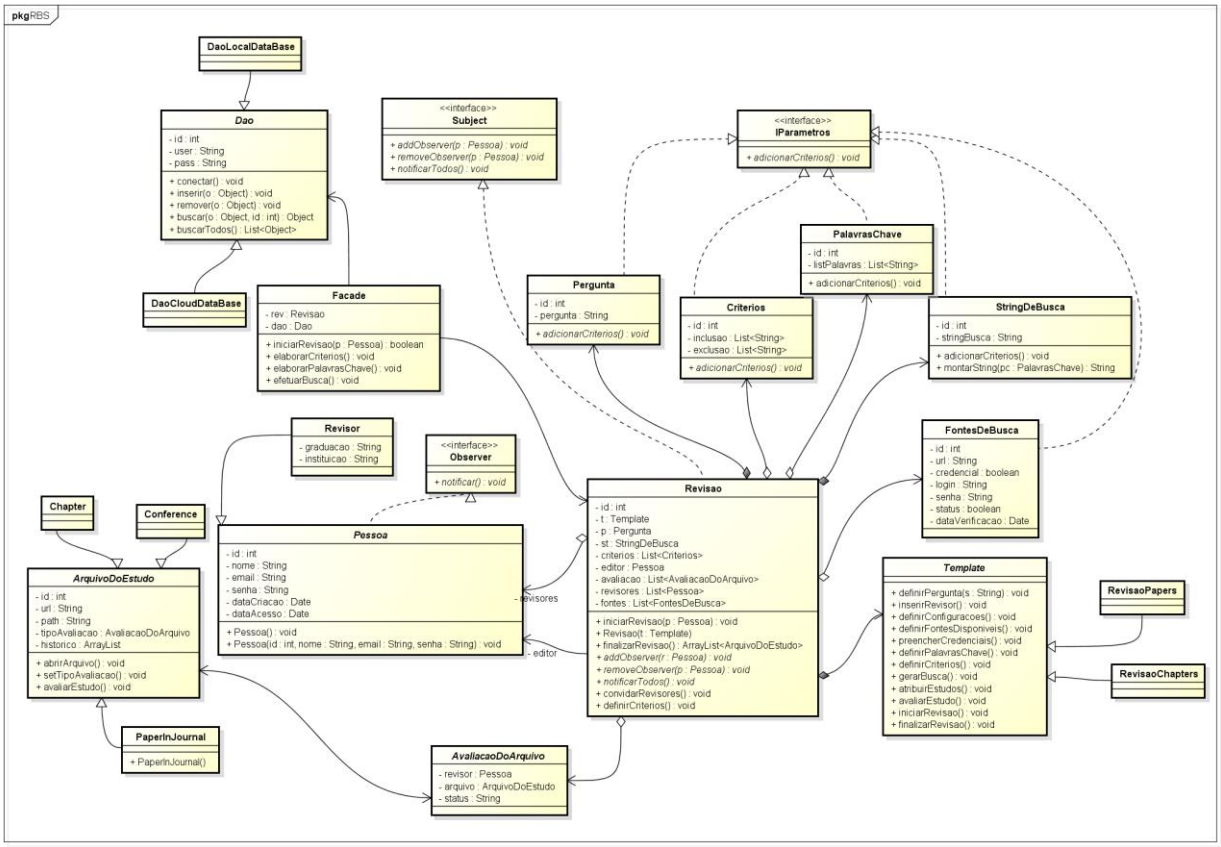

Figura 14 - Diagrama de Classes do Sistema de Revisão Sistemática Reestruturado
Fonte: Autoria Própria.

A nova versão do modelo de classes é apresentada na Figura 15, demonstrando a importância da utilização dos conceitos de POO e princípios de projeto, os quais tornam o modelo flexível e aberto para adições de novas demandas com menores impactos. As classes abstratas `Pessoa`, `ArquivoDoEstudo`,

`AvaliacaoDoArquivo`, `Template`, `Dao` representam tipos generalizados de objetos.

- `Pessoa`: representa uma abstração generalizada, neste caso existe apenas uma classe que extende esta classe abstrata, que é a classe `Revisor`, esta dependência permite que sejam inseridos novos tipos de objetos que sejam dependentes de `Pessoa` apenas criando novas classes e exercendo a sua dependência para a classe abstrata;
- *`ArquivoDoEstudo`*: esta classe abstrata representa uma forma generalizada dos arquivos que serão avaliados e armazenados, neste caso existem três classes concretas que extendem `ArquivoDoEstudo`, permitindo desta maneira que sejam classificados os estudos de acordo com suas características e sejam inseridos novos tipos de estudos, bastando apenas criar novas classes e relacioná-las com a classe abstrata;
- *`AvaliacaoDoArquivo`*: representa os diferentes tipos de avaliação que um determinado estudo pode ter, permitindo que sejam inseridos novos tipos de avaliações de arquivos, sem causar impactos no código já existente na aplicação;
- *`Template`*: esta classe abstrata é parte da aplicação de padrões deprojeto, ao qual será detalhada futuramente e define uma estrutura para diferentes tipos de revisões bibliográficas sistemáticas, permitindo a criação de *templates* para revisões específicas como o exemplos das classes concretas `RevisãoPapers` e `RevisaoChapters`;
- `Dao`: representa a abstração dos meios de armazenamento que o *software* pode vir a ter futuramente, dando assim uma flexibilidade para que o sistema receba atualizações de novas funcionalidades causando o menor impacto possível no código já existente.

As interfaces foram incluídas visando a aplicação de padrões de projeto (`Observer` e `Subject`) e a aplicação de métodos que determinadas classes devem possuir (`IParâmetros`).

- `Observer`: é uma das interfaces de aplicação do padrão *Observer*, a relação com esta interface representa que o objeto concreto que implementa esta interface é um observador e permitindo que toda alteração realizada pelo mesmo, possa ser replicada à outros observadores;

- `Subject`: é a *Interface* que representa o objeto concreto observado, o objeto que implementa esta interface pode relacionar novos observadores, removê-los e notificar todos os objetos observadores quando houver alguma mudança em seu estado;
- `IParametros`: *interface* que representa uma ação comum à todas as classes que representem parâmetros de entrada do processo de revisão sistemática.

Desta maneira, fica evidente a praticidade de manutenção e incrementação do novo modelo, permitindo que haja a adição de novos objetos concretos dos tipos como `ArquivoDoEstudo`, `FontesDeBusca`, `Pessoa`, `Template`, sem que haja a necessidade de replicar estas adições em outras classes, por exemplo, para inserir um novo tipo de `ArquivoDoEstudo`, basta criar uma nova classe concreta e relacioná-la com a classe abstrata, tornando assim evidente as vantagens de se utilizar as técnicas de POO e princípios de projeto.

O diagrama de classes correspondente ao módulo de *chat* é apresentado na Figura 16, de modo que é possível observar a aplicação do padrão de projeto *Template* para a criação dos métodos do servidor, permitindo assim que se houverem novos tipos de servidores estes possam ser implementados sem maiores impactos no sistema.

A classe `UsuarioChat` será encarregada de armazenar as mensagens de cada usuário, já a classe `Servidor` irá representar as informações e conexão com um servidor, realizando as operações de gerenciamento e envio de mensagens.

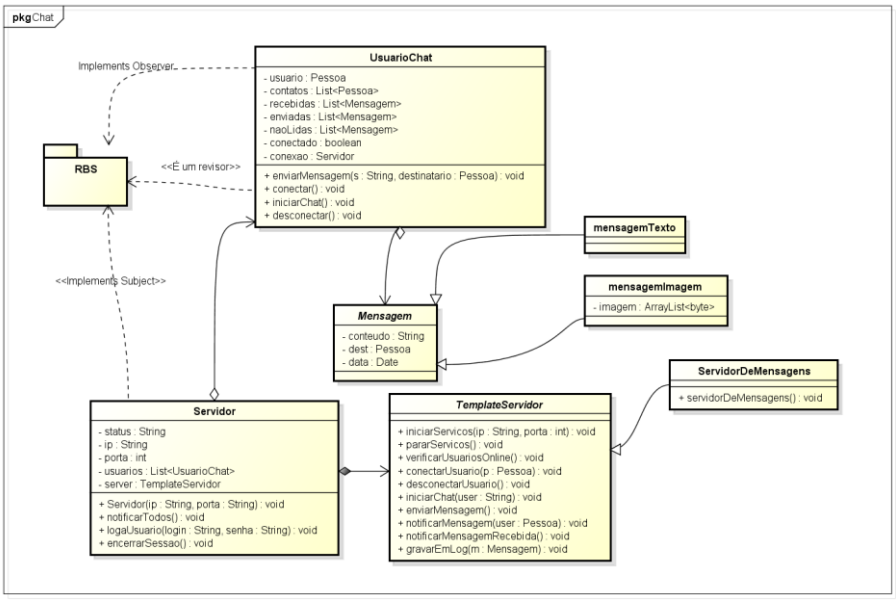

Figura 15 - Diagrama de Classes do Módulo de Chat
Fonte: Autoria Própria.

O desenvolvimento destes diagramas de classes possui foco na aplicação de conceitos de POO e padrões, buscando uma solução flexível e robusta, que permita a escalabidade com o menor impacto possível.

O diagrama de sequência apresentado na Figura 17, representa o Caso de Uso F3 Criar Revisões (Tabela 5), demonstrando a aplicação da camada do padrão *facade*, que apresenta aos clientes uma única *interface* de comunicação com o sistema, oferecendo métodos simples, dispensando um alto nível de conhecimento e acesso à lógica de negócios do sistema, sendo o responsável por efetuar as chamadas de métodos com outros objetos do sistema.

Os ciclos de interação são determinados por meio do padrão *template*, que define uma estrutura pré-definida de passos a serem seguidos durante a construção de uma revisão sistemática, baseando-se no tipo de revisão criada.

O padrão *observer* é utilizado para inserir objetos do tipo `Pessoa` à lista de observadores do objeto observado em questão (`Revisão`) e notificá-los sobre a criação ou alteração do estado deste objeto.

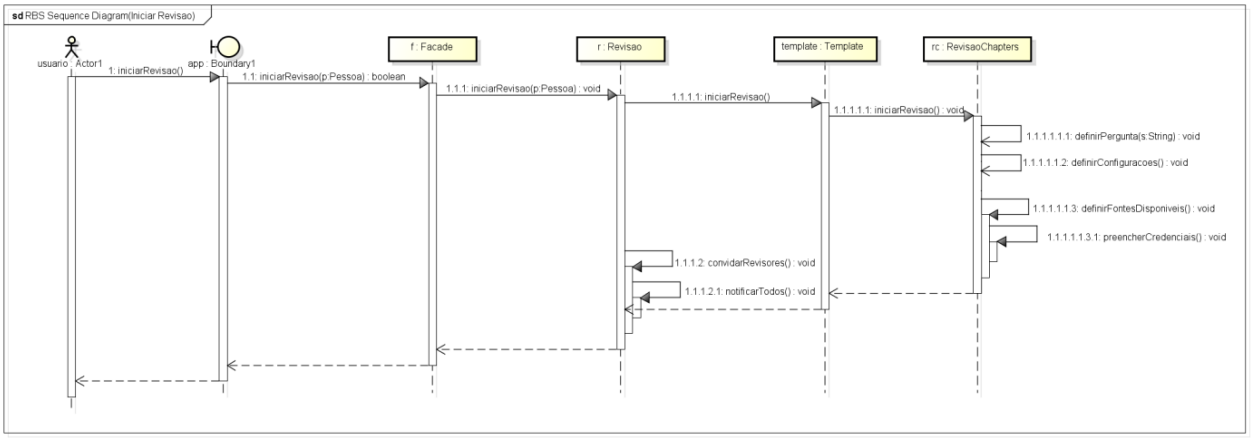

Figura 16 - Diagrama de Sequência sobre o caso de uso F3 Criar Revisão
Fonte: Autoria Própria.

A Figura 18 apresenta o diagrama de sequência referente ao caso de uso F13 – Iniciar Chat, de modo a apresentar a interação com o a classe `servidor` e como o mesmo gerencia os serviços disponibilizados.

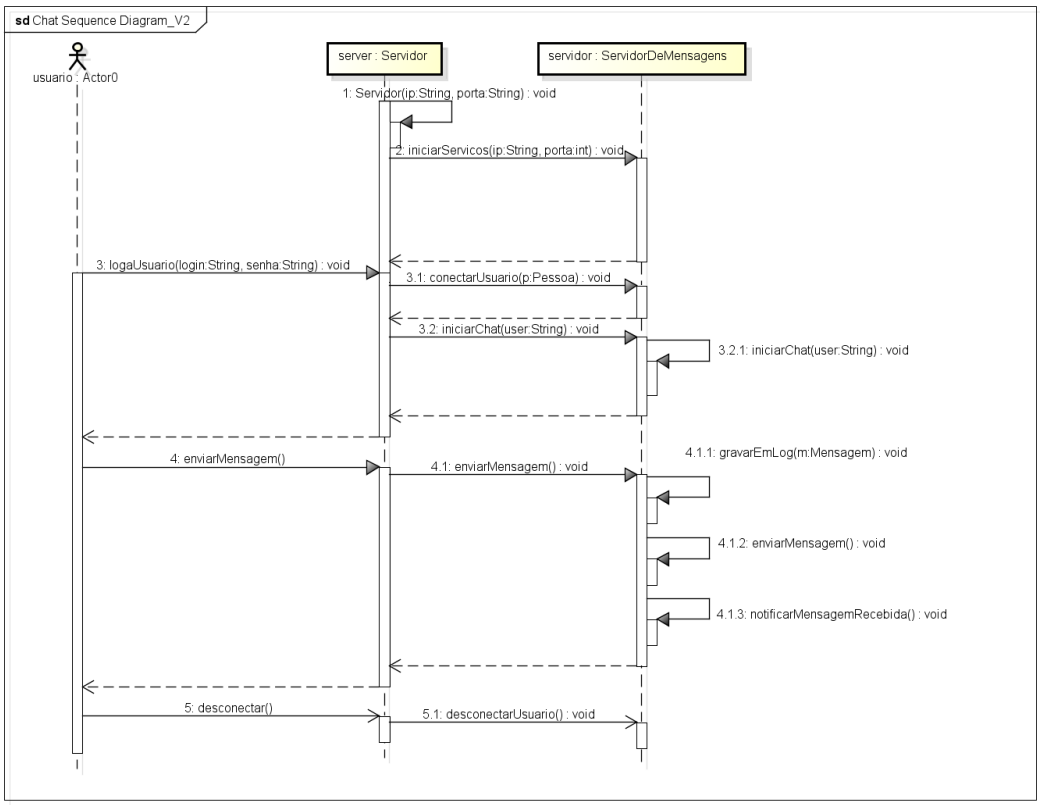

Figura 17 - Diagrama de Sequência do Caso de Uso F13 - Criar Revisão
Fonte: Autoria Própria.

O diagrama de atividades apresentado na Figura 19 demonstra o funcionamento geral do módulo de revisão sistemática. É divido em três raias (*swim lanes*) que representam as fronteiras de cada elemento que repesenta o processo, apresentando as atividades relacionadas à cada elementos. `Cliente` é que deve gerar as solicitações para as atividades, `App` é a interface de comunicação (neste caso, uma interface *web* como proposto nos requisitos do projeto) e o `Sistema RBS` é o qual se encontra toda a lógica de negócio, que realiza todas as atividades relacionadas ao processo de revisão sistemática.

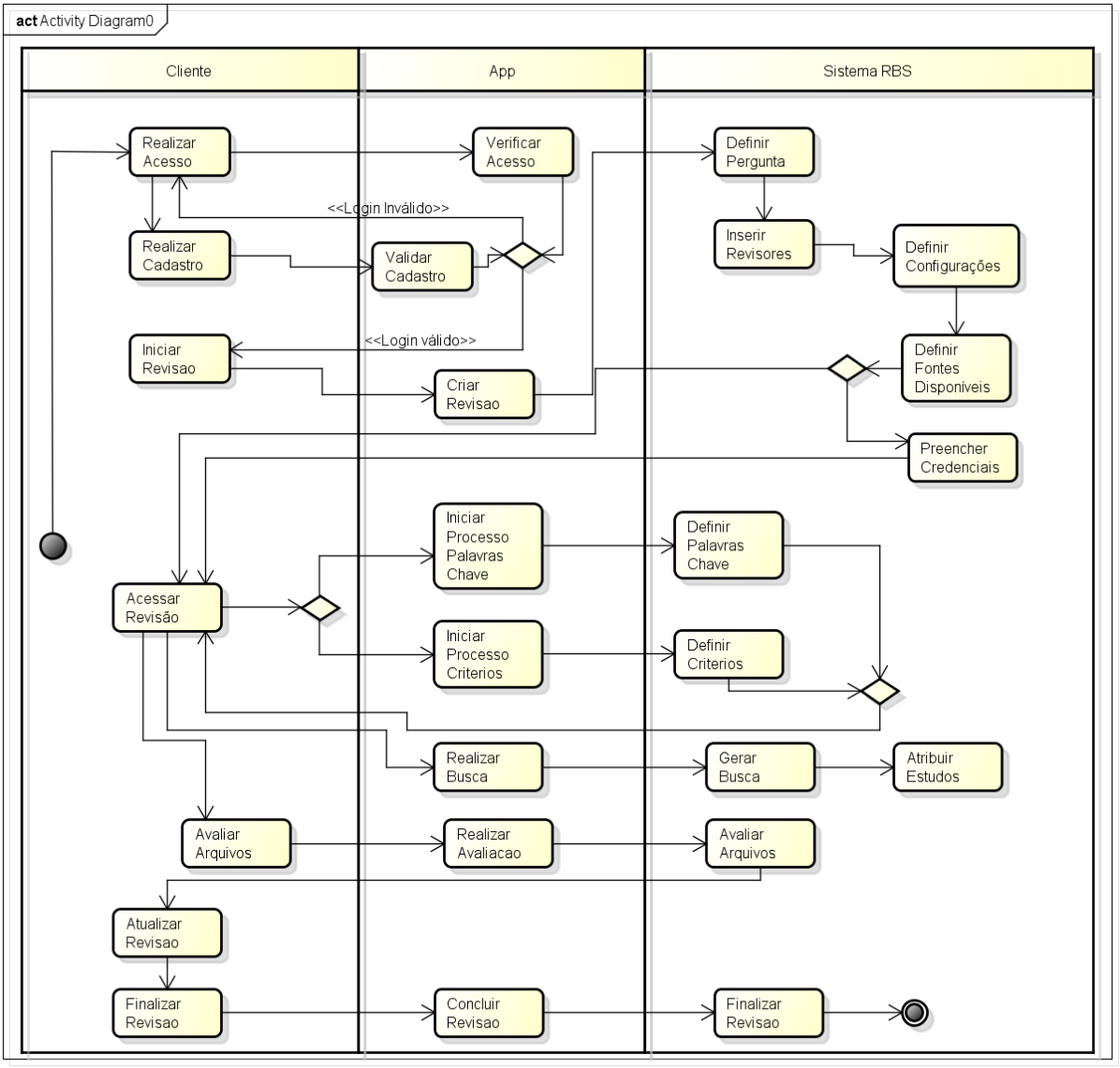

Figura 18 - Diagrama de Atividades da Ferramenta de Revisão Sistemática
Fonte: Autoria Própria.

4.2.1.1 Problemas encontrados

No diagrama de classes apresentado, havia um alto índice de coesão entre classes, pois as dependências ocorrem diretamente entre as classes concretas, gerando deste modo um modelo que dificulta a inserção de novas funcionalidades, como novos tipos de revisores, arquivos, sem que haja um grande impacto nas classes aos quais estão ligados e acrescendo um alto custo de manutenção.

Um *software* mal projetado na etapa de análise (compreenda-se etapas de levantamento de requisitos, análise orientada a objetos e documentação), possui pouca expectativa de vida, tendendo a tornar-se algo estático, que dificilmente estará apto a receber novas funcionalidades, ou ainda caso sejam inclusas novas funcionalidades, o impacto no nível de complexidade será cada vez maior até determinado ponto em que torna-se totalmente inviável e o *software* tem de ser descartado ou desenvolvido novamente.

Prevendo o custo de tempo e o esforço empregado para reestruturar a organização das classes, optou-se por conceber um novo diagrama, aplicando os conceitos de POO e padrões de projeto, alinhando os conceitos de boas práticas. Na Figura 14 é apresentado o diagrama proposto incialmente para o desenvolvimento da ferramenta.

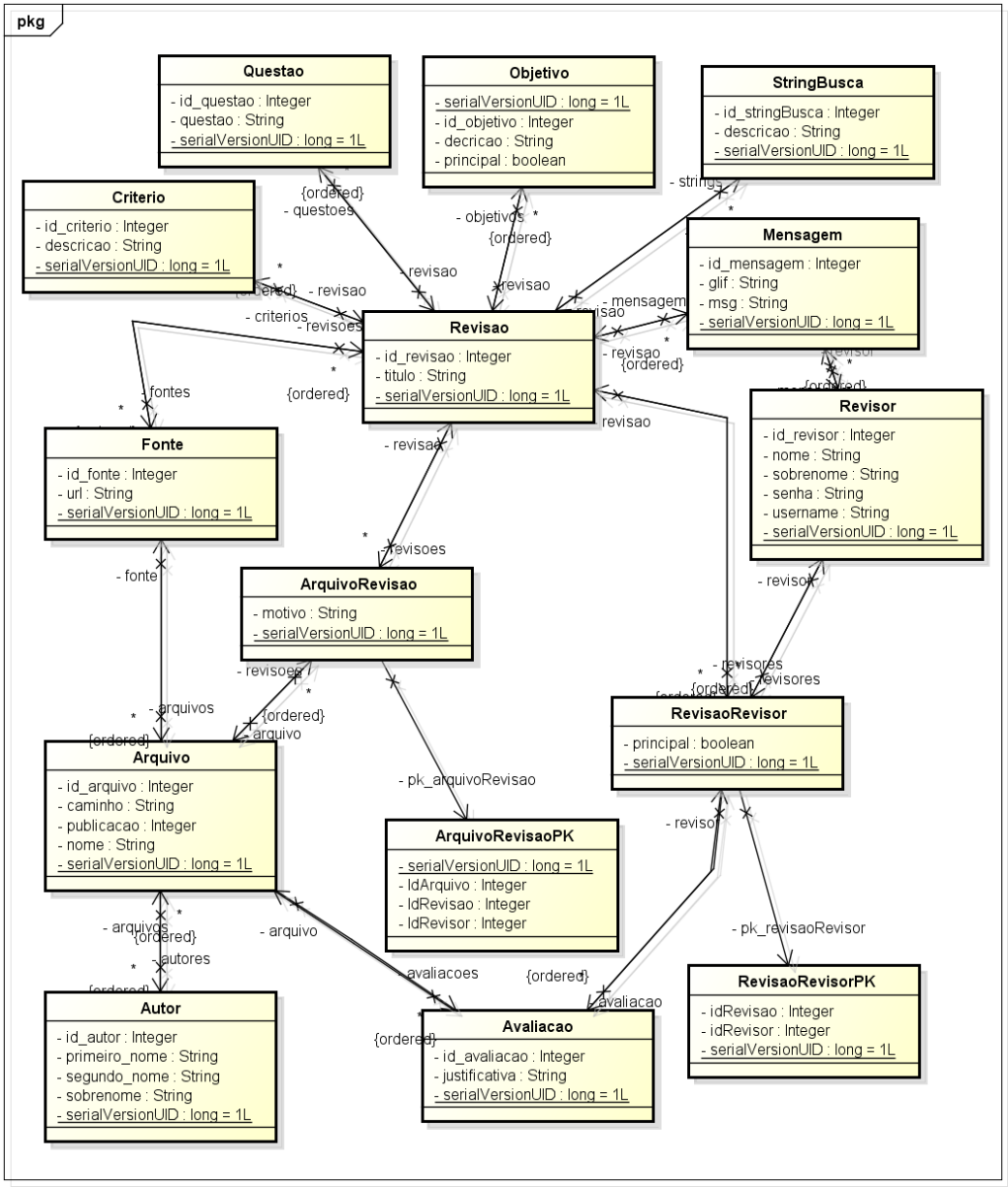

Figura 19 - Diagrama de Classes da ferramenta de Revisão Sistemática em sua primeira versão
Fonte: Ferreira(2014).

4.2.1.2 Proposta de melhorias

O novo diagrama de classes é apresentado com o foco na correção do modelo com a aplicação dos conceitos de POO, princípios de projeto e padrões de projeto.

Utilizando interfaces e classes abstratas para representarem super tipos de classes, fazendo assim com que as classes concretas não dependam diretamente entre si, facilitando assim a expansão e possível correção do software em alguma necessidade. Os modificadores de acessos `Gets` e `Sets` não estão representados no diagrama de classes por fins de visualização e compreendimento.

Os padrões de projeto foram inseridos visando dar flexibilidade, encapsulamento, suprir demandas da aplicação e agilidade no processo de manutenções futuras, os padrões aplicados neste *software* foram *Observer, Facade* e *Template Method*.

- *Observer*: define a interação de notificação entre um objeto que é observado e por um ou mais objetos observadores, este padrão é aplicado quando objetos necessitam recuperar informações de um determinado objeto quando este muda seu estado, este padrão é aplicado entre as classes `Revisao` (Observado) e `Pessoa` (observadores). A classe `Revisao` implementa a interface `Subject` que lhe fornecerá os métodos para controlar os observadores e notificá-los;
- *Facade*: o padrão *facade* visa oferecer um único meio de comunicação entre um determinado cliente externo e o sistema que se está utilizando, fazendo assim com que o cliente conheça apenas a fachada do sistema e esta fachada que será a responsável por interagir diretamente com o sistema. A classe `Facade` é a responsável por manter este controle de acesso, fornecendo métodos simplificados para quem a acessa.
- *Template Method*: este padrão define uma estrutura de template de passos a serem seguidos, neste caso a classe abstrata *Template* apresenta todos os métodos de uma revisão, fornecendo estes métodos para cada classe concreta implementá-lo como houver necessidade.

A classe `Revisao` pode ser considerada a classe central do diagrama de classes, pois nela devem constar os conteúdos referentes ao processo de revisão sistemática. Nota-se a utilização de objetos de classes abstratas na declaração de seus atributos, esta utilização permite uma generalização de tipos, oferecendo uma maior flexibilidade ao utilizar o sistema sem replicação de código, por exemplo o atributo `avaliacao`, é uma lista de objetos do tipo `AvaliacaoDoArquivo`, permitindo assim que tenham diversos objetos concretos que extendem a classe

abstrata *AvaliacaoDoArquivo*, evitando assim que se tenha a necessidade de utilizar uma lista específica para cada tipo de avaliação.

4.3 CODIFICAÇÃO

Para apresentar na prática os conhecimentos aplicados na etapa de análise, foram desenvolvidas as classes e métodos utilizando a *IDE NetBeans*, elaborando apenas saídas em console para demonstrar o funcionamento baseado no que foi proposto na etapa de análises. Nos códigos 1 a 3 são apresentados os métodos relacionados à classe `Revisão`, demonstrando assim como estão sendo efetuadas as chamadas de métodos e saídas em *console*, pode-se observar as chamadas de métodos `notificarTodos`, `addObserver`, `removeObserver`, que apresentam a implementação do padrão *observer*, implementando a *interface* `Subject`, a classe `Revisão` torna-se um objeto observado, gerenciando seus observadores e notificando-os por meio dos métodos listados.

Código 1 - Métodos da Classe Revisao Parte 1

```
53      public void iniciarRevisao(Pessoa editor) {
54          this.editor = editor;
55          t.iniciarRevisao();
56          String nome = JOptionPane.showInputDialog("Informe o nome do revisor que
        será inserito");
57          Revisor revisor = new Revisor(nome);
58          this.convidarRevisores(revisor);
59          this.notificarTodos();
60
61      }
62
63      public void definirCriterios(){
64          System.out.println("Delegando Definição de critérios pela estrutura do
        template.............");
65          t.definirCriterios();
66          this.notificarTodos();
67      }
68
69      public void definirPalavrasChave(){
70          System.out.println("Delegando Definição das palavras-chave pela estrutura de
        template...");
```

```
71          t.definirPalavrasChave();
72          this.notificarTodos();
73      }
74
75      public void efetuarBusca(){
76          System.out.println("Delegando busca para a estrutura template");
77          t.gerarBusca(st, fontes);
78      }
```

Fonte: Autoria Própria.

Código 2 - Métodos da Classe Revisao Parte 2

```
80      public void listarStatusTodasAvaliacoes() {
81          for (AvaliacaoDoArquivo a : avaliacao) {
82              System.out.println(a);
83          }
84      }
85
86      public String efetuarUmaAvaliacao(Pessoa revisor, ArquivoDoEstudo arquivo) {
87          System.out.println("Meu querido avaliador " + revisor.getNome()
88                  + "\nQual sua avaliação quanto ao arquivo " + arquivo.getPath() +
    "?");
89          String st = JOptionPane.showInputDialog(null, "Informe avaliação: ");
90          return st;
91      }
92
93      public void addAvaliacao(Pessoa revisor, ArquivoDoEstudo arquivo, String s) {
94          AvaliacaoDoArquivo a = new AvaliacaoDoArquivo(revisor, arquivo, s);
95          avaliacao.add(a);
96          System.out.println("Uma avaliação foi adicionada ...");
97      }
98
99      public boolean convidarRevisores(Pessoa convidado) {
100         if (convidado != null) {
101             revisores.add(convidado);
102             this.addObserver(convidado);
103             this.notificarTodos();
104             System.out.println("Um revisor foi adicionado ...");
105             return true;
106         }else{
107             return false;
108         }
109     }
```

Fonte: Autoria Própria.

No Código 3 são apresentados os métodos para a inclusão de novos revisores e os métodos relacionados à interface `Subject` (padrão de projeto *Observer*), que são responsáveis por gerenciar os objetos observadores.

Código 3 - Métodos da Classe Revisao Parte 3

```
110     @Override
111     public void addObserver(Pessoa p) {
112         System.out.println("Inserindo novo observador " + p.getNome() + ".....");
113         this.notificarTodos();
114     }
115
116     @Override
117     public void removeObserver(Pessoa p) {
118         System.out.println("Removendo " + p.getNome() + "....");
119         this.notificarTodos();
120     }
121
122     @Override
123     public void notificarTodos() {
124         System.out.println("Notificando Todos os Observadores.....");
125     }
```

Fonte: Autoria Própria.

Seguindo a ordem das chamadas das funções utilizando a classe `Facade`, é responsável por efetuar a chamada do método `iniciarRevisão` que possui um parâmetro de entrada do tipo `Pessoa`, demonstrando deste modo o funcionamento do padrão de projeto *Facade*, encapsulando a lógica de negócios dos clientes que a acessam. O Código 4 apresenta todo o conteúdo da classe `Facade`, apresentando os seus métodos.

Código 4 - Classe Facade

```
15  public class Facade {
16      Pessoa p;
17      Revisao r;
18
19      public Facade() {
20          ////this.r = r
21          r = new Revisao();
22      }
23
24      public void iniciarRevisao(Pessoa p) {
25          System.out.println("Iniciando Revisão pelo sistema de interface do Facade");
26          r.iniciarRevisao(p);
27      }
28
29      public void elaborarCriterios(){
```

```
30          System.out.println("Elaborando a etapa de definição de critérios pelo
        sistema de interface facade");
31          r.definirCriterios();
32      }
33
34      public void elaborarPalavrasChave(){
35          System.out.println("Elaborando etapa de escolha da palavras-chave pelo
        sistema de interface facade");
36          r.definirPalavrasChave();
37      }
38
39      public void elaborarBusca(){
40          System.out.println("Elaborando o processo de montagem da busca pelo sistema
        de interface facade.....");
41          r.efetuarBusca();
42          r.notificarTodos();
43      }
44  }
```

Fonte: Autoria Própria.

A classe `facade` é reponsável por criar novos objetos `Revisao` e realizar as chamadas de seus respectivos métodos, isolando assim a aplicação da visão do cliente, sendo então responsável por encaminhar as solicitações por meio de chamadas de métodos. No Código 5 é apresentado a declaração dos m'etodos construtores da classe `Revisao`, que demonstra a utilização de um objeto `Template` que irá definir os passos a serem seguidos para construir a revisão especificamente.

Código 5 - Métodos Construtores da classe Revisor

```
39      public Revisao() {
40          this.t = new RevisaoPapers();
41          this.criterios = new ArrayList<>();
42          this.avaliacao = new ArrayList<>();
43          this.revisores = new ArrayList<>();
44          this.fontes = new ArrayList<>();
45          //inseridos valores para tentar construir o fluxo da revisao
46
47      }
48
49      public Revisao(Template t) {
50          this.t = t;
51      }
```

Fonte: Autoria Própria.

A utilização do padrão de projeto *Template Method* implica que a classe `Revisao` possua um atributo do tipo `Template`(abstrato), permitindo que a construção dos métodos da revisão seja realizada de acordo com o que está proposto na classe concreta que extende a classe abstrata `Template`, demonstrando assim a aplicação do padrão proposto no modelo de clases. O Código 6 apresenta a classe `Main` que irá realizar a execução dos códigos para demonstrar a aplicação dos conceitos definidos na etapa de análise.

Código 6 - Classe Main que irá executar os métodos para apresentar o funcionamento dos conceitos aplicados

```
17  public class Main {
18
19      public static void main(String[] args) {
20         ArquivoDoEstudo artigoMeditec = new PaperInJournal();
21         artigoMeditec.setPath("c:\\windows\\teste.txt");
22          /* Instanciando Revisores   */
23         Pessoa juliano = new Revisor("Juliano");
24         Pessoa ivan = new Revisor("Ivan");
25         Pessoa arnaldo = new Revisor("Arnaldo");
26         Pessoa rodrigo = new Revisor("Rodrigo");
27         Revisao revisao = new Revisao();
28          //revisao.iniciarRevisao(ivan);
29
30          System.out.println(".........a partir deste ponto temos o sistema de
    interface agindo...");
31          Facade f = new Facade();
32          f.iniciarRevisao(ivan);
33          f.elaborarCriterios();
34          f.elaborarPalavrasChave();
35          f.elaborarBusca();
36      }
```

Fonte: Autoria Própria.

A saída resultante desta execução encontra-se na Figura 20, demonstrando que utilizando o *facade* houve a comunicação e execução de todos os métodos relacionados à revisão.

```
Updating property file: C:\Users\Ivan\Documents\NetBeansProjects\rbs2.1\build\built-jar.properties
Compiling 1 source file to C:\Users\Ivan\Documents\NetBeansProjects\rbs2.1\build\classes
compile:
run:
..........a partir deste ponto temos o sistema de interface agindo...
Iniciando Revisão pelo sistema de interface do Facade
Definindo Configurações.....
Definindo pergunta.......
Pergunta Teste
Escolhendo Fontes de Dados Disponíveis....
Preenchendo Credenciais para Acesso as fontes....
Notificando Todos os Observadores.....
Um revisor foi adicionado ...
Notificando Todos os Observadores.....
Elaborando a etapa de definição de critérios pelo sistema de interface facade
Delegando Definição de critérios pela estrutura do template..............
Definindo Critérios de Inclusão e Exclusão....
Notificando Todos os Observadores.....
Elaborando etapa de escolha da palavras-chave pelo sistema de interface facade
Delegando Definição das palavras-chave pela estrutura de template...
Definindo Palavras Chave....
Notificando Todos os Observadores.....
Elaborando o processo de montagem da busca pelo sistema de interface facade.....
Delegando busca para a estrutura template
Gerando Busca de Estudos......
Atrbiuindo Estudos aos revisores.....
Notificando Todos os Observadores.....
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

Figura 20 - Console de Saída da Execução do arquivo Main
Fonte: Autoria Própria.

Demonstrando a funcionalidade da aplicação dos conceitos propostos para o desenvolvimento do novo modelo da ferramenta, disponibilizando uma única interface de comunicação do sistema utilizando o *Facade*, definindo uma estrutura base para a elaboração de uma revisão utilizando *Template Method* e notificando os revisores que estão relacionados à uma revisão utilizando o padrão *Observer*, todo este conjunto de funcionalidades aplicadas com padrões de projeto permitem que o modelo proposto para o desenvolvimento desta ferramenta tenha condições de ser um modelo funcional e incremental.

4.4 PROTÓTIPOS

Os protótipos incrementam a etapa de análise por fornecerem um conteúdo visual que transmite a idéia aproximada do *layout* final, permitindo assim que seja contemplado o modelo final de apresentação da ferramenta, auxiliando na identificação de soluções e melhorias sem a necessidade de se desenvolver

efetivamente a camada visual da ferramenta. São apresentados os *layouts* desenvolvidos para este *software* das Figuras 21 à 25. A Figura 21 apresenta o *layout* da página incial da ferramenta.

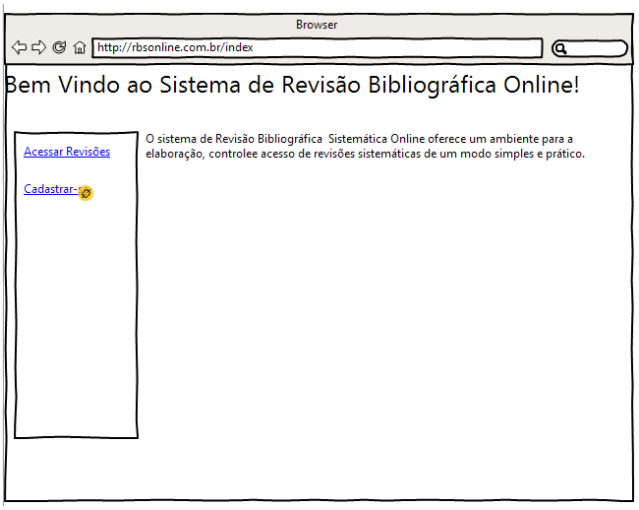

Figura 21 - Protótipo da Página Index da Ferramenta de Revisão Sistemática
Fonte: Autoria Própria.

A Figura 22 apresenta o *layout* da página de cadastro de novos usuários correspondente ao Quadro 3 – Caso de Uso F1 Cadastrar Usuário.

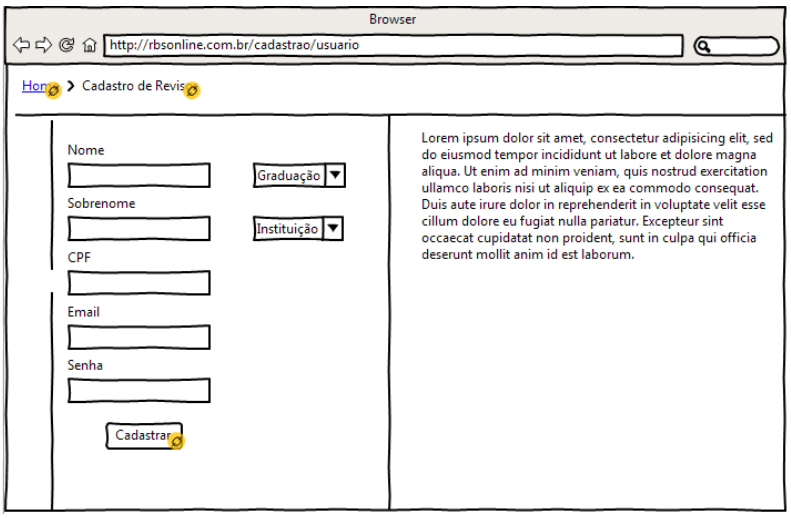

Figura 22 - Layout da Página de Cadastro de Usuários
Fonte: Autoria Própria.

A Figura 23 apresenta o cadastro de uma revisão sistemática baseado no Quadro 5 – Requisito F3 – Cadastrar Revisão, demonstrando a utilização do padrão

Template Method que visa oferecer um roteiro de cadastro para cada tipo de revisão que seja implementado no sistema, neste caso, criou-se um tipo genérico de revisão sistemática para apresnetação do *layout.* Também é apresentada a utilização do padrão *observer* (ícone de Notificações), no qual serão apresentadas as atualizações referentes à cada atualização do estado do processo de revisão, notificando às informações relacionadas à cada usuário e a área de troca de mensagens utilizando o módulo de *chat* proposto, por meio deste serão disponibilizados os contatos de cada usuário, com seus respectivos status, um campo para pesquisas à partir da lista de contatos e a opção para a inserção de novos contatos, mediante aprovação..

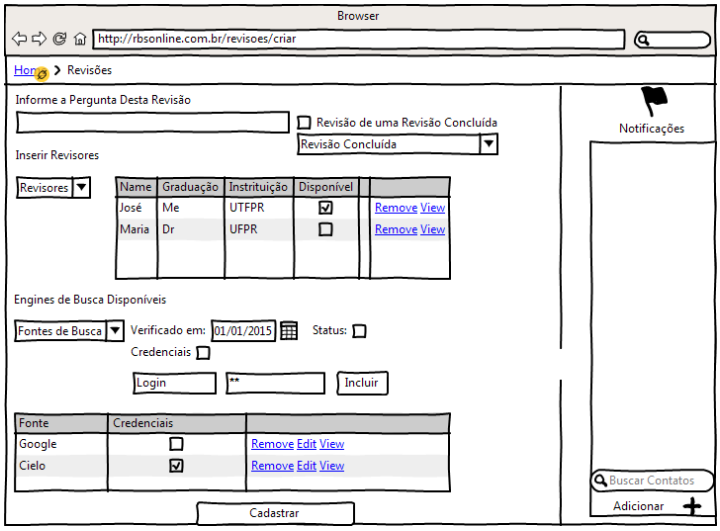

Figura 23 - Layout da página de criação de uma nova revisão sistemáica
Fonte: Autoria Própria.

As Figuras 25 e 26 apresentam os processos de seleção das palavras-chaves e critérios, o foco é que esta atividade seja realizada com todos os revisores envolvidos no processo de revisão, pois são etapas que necessitam a colaboração em tempo real entre os envolvidos. Na Figura 24 é apresentado o *layout* da tela de seleção das palavras-chave, baseados no caso de uso do Quadro 9 – Requisito Funcional F7(Definir Palavras Chave).

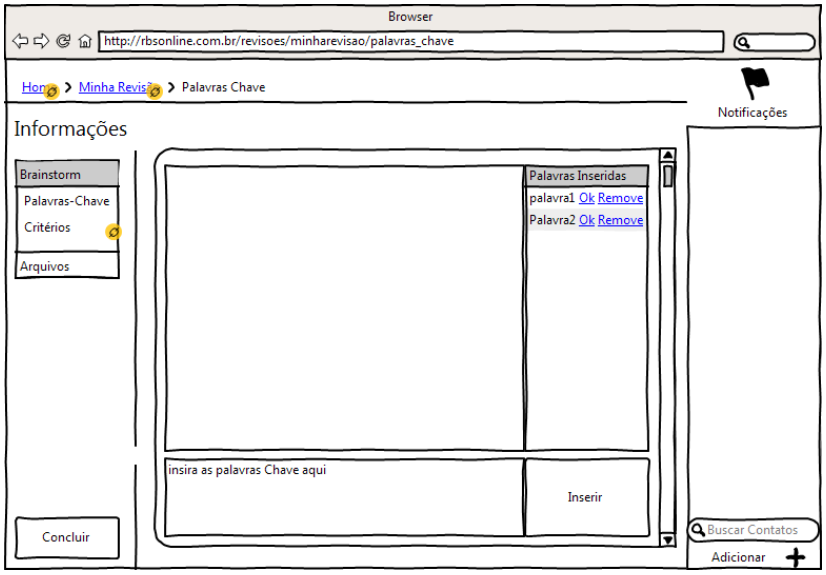

Figura 24 - Layout da página de definição das palavras-chave
Fonte: Autoria Própria.

Na Figura 25 é apresentado o *layout* da página que representa o processo de escolha dos critérios de inclusão ou exclusão, desenvolvid à partir do Quadro 10 – Requisito Funcional F8 (Definir Critérios).

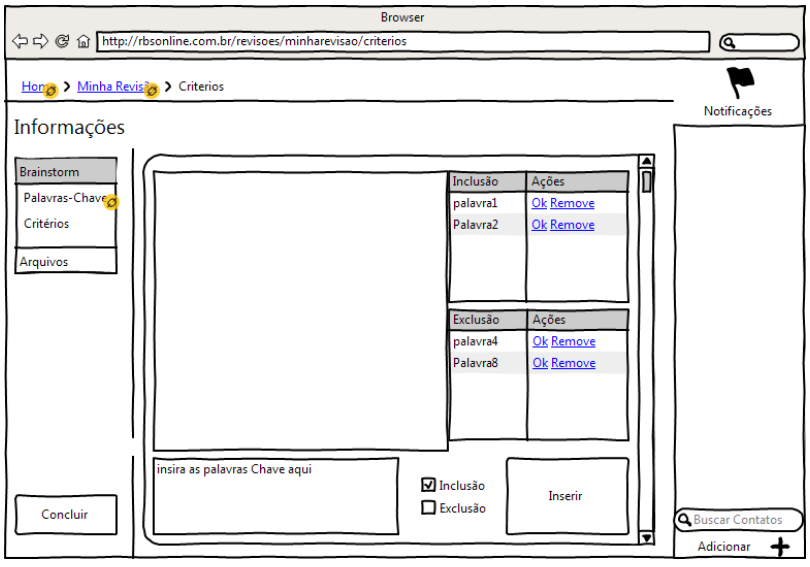

Figura 25 - Layout da página de cadastro dos critérios
Fonte: Autoria Própria.

Protótipos são um grande auxílio para a tomada de decisão, pois fornecem ao cliente uma perspectiva essencial da apresentação final do *software*, facilitando assim as tomadas de decisão, identificação de requisitos adicionais ou ainda a

correção de problemas relacionados à interação da ferramenta com os usuários, por outro lado, auxilia os desenvolvedores pois oferece uma base visual para a construção da camada de apresentação do *software*, tornando o processo de criação desta camada eficiente, pois não existe a necessidade de investir muito tempo modelando toda a camada de apresentação à partir do nada.

5 CONSIDERAÇÕES FINAIS

São apresentadas as considerações à respeito do trabalho desenvolvido.

5.1 CONCLUSÃO

O desenvolvimento de um *software* não é algo que possa ser elaborado sem respeitar alguns fluxos de elaboração, omitindo etapas importantes que auxiliem a identifiar necessidades e problemas muito antes do seu desenvolvimento efetivo. As consequências deste desenvolvimento precário são os custos de manutenção e reestruturação, pois perdem-se recursos, destes, dois que jamais devem ser menosprezados, o tempo e a confiabilidade. Um produto final mal projetado e desevolvido só tende a oferecer problemas para todos os envolvidos desde à sua concepção até os envolvidos na utilização do produto.

Basear a concepção e o desenvolvimento em conhecimentos, técnicas e conceitos, não pode ser considerado uma extravagância somente para pequenos ou projetos de grande importância, é algo fundamental, pois estes conhecimentos estão disponíveis justamente para auxiliar o desenvolvimento de produtos com qualidade.

O esforço empregado para a correção de um *software* que já está em fase de desenvolvimento pode ser comparado ao esforço de desenvolver um novo projeto, requerendo um custo muito maior pois é necessário realizar um estudo inicial para verificar em qual ponto encontra-se o *software* suas delimitações e incoerências.

A reestruturação deste projeto permitiu a criação de uma documentação adequada, permitindo deste modo que o projeto seja continuado ou aprimorado sem um esforço desncessário para o entendimento do que foi desenvolvido. A aplicação das técnicas de POO e conceitos de padrões de projeto e princípios de projetos forneceram à base sólida que todo *software* necessita, uma base documentada e que permita a inclusão de novas funcionalidades minimizando os impactos, tornando assim uma ferramente flexível que seja capaz de suportar as mudanças de suas demandas, criação de novos serviços mantendo sempre a confiabilidade.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

A reestruturação desta ferramenta visa fornecer uma base para que este projeto possa receber a continuidade em seu desenvolvimento, oferecendo a documentação e elaboração do modelo de classes, permitindo assim, que por meio da aplicação de conceitos de POO, padrões de projeto e princípios de projeto, este projeto tenha condições de seguir futuramente para a etapa de desenvolvimento com sua base fundamentada, tornando a ferramenta escalável e flexível.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, Everton Coimbra de et al. **Padrões de Projeto em Aplicações Web**. Florianópolis: Visual Books, 2013. 140 p.

BAX, Marcelo Peixoto. **Introdução as Linguagens de Marcas**. Ciência da Informação, Brasília, v. 30, n. 1, p.32-38, 2001. Quadrimestral. Disponível em: <http://www.scielo.br/pdf/ci/v30n1/a05v30n1.pdf>. Acesso em: 27 ago. 2014.

CONFORTO, Edivandro Carlos; AMARAL, Daniel Capaldo; SILVA, Sérgio Luis da. **Roteiro para revisão bibliográfica sistemática: aplicação no desenvolvimento de produtos e gerenciamento de projetos**. In: Congresso Brasileiro de Gestão de Desenvolvimento de Produto, 8., 2011, Porto Alegre. Anais... . Porto Alegre: Cbgdp, 2011. v. 1, p. 1 - 12.

FERREIRA, Ana Caroline. **FERRAMENTA WEB COM HTML5 E JAVA PARA O PROCESSO DE REVISÃO SISTEMÁTICA DA LITERATURA**. 2014. 57 f. TCC (Graduação) - Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Medianeira, 2014.

FREEMAN, Eric; FREEMAN, Elizabeth. **Use a Cabeça!: Padrões de Projeto**. 2. ed. Rio de Janeiro: Alta Books, 2007. 496 p.

GAMMA, Erich et al. **Padrões de Projeto:** Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. 368 p.

GOULART, Reane Franco. **UML e a Ferramenta Astah**. Ituiutaba: Instituto Federal de Educação, Ciência e Tecnologia, 2007. 21 slides, color. Disponível em: <http://slideplayer.com.br/slide/4127954/>. Acesso em: 28 out. 2015.

GUEDES, Gilleanes T. A.. **UML 2: Guia Prático**. São Paulo: Novatec, 2007. 176 p. Disponível em: <http://www.novateceditora.com.br/livros/uml2/capitulo9788575221457.pdf>. Acesso em: 17 out. 2015.

GUERRA, Eduardo. **Design Patterns com Java:** Projeto orientado a objetos guiado por padrões. São Paulo: Casa do Código, 2012. 305 p.

KON, Fabio. **Uma Visão Geral de UML**. São Paulo: Ime/usp, 2005. 65 slides, color. Baseado em Slides de Kendall V. Scott. Disponível em: <https://www.ime.usp.br/~kon/presentations/UMLIntro.pdf>. Acesso em: 16 out. 2015.

LEITE, Mário; RAHAL JÚNIOR, Nelson Abu Sanra. **Programação Orientada a Objeto: uma abordagem didática: Object-Oriented Programming**: **a didatic presentation**.

Disponível em: <http://www.ccuec.unicamp.br/revista/infotec/artigos/leite_rahal.html>. Acesso em: 16 out. 2015.

LOPES, Ana Lúcia Mendes. **Revisão Sistemática de literatura e metassíntese qualitativa: Considerações sobre sua aplicação na pesquisa em enfermagem**. 2008. 778 f. Dissertação (Mestrado) - Curso de Enfermagem em Saúde Coletiva, Universidade de São Paulo, São Paulo, 2008. Cap. 4. Disponível em: < http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-07072008000400020>. Acesso em: 28 ago. 2014.

MENDES, Douglas Rocha. **Programação Java: com ênfase em Orientação a Objetos**. São Paulo: Novatec, 2009. 456 p. Disponível em: <http://novatec.com.br/livros/javaoo/capitulo9788575221761.pdf>. Acesso em: 16 out. 2015.

RF, Sampaio; MC, Mancini. **Estudos de revisão sistemática: Um guia para síntese criteriosa da evidência científica**. Revista Brasileira de Fisioterapia, Belo Horizonte, v. 11, n. 11, p.83-89, fev. 2007.

SANTOS, Edinei Fabiano Fedel dos; LEMOS, Léia Lúcia de. **Desenvolvimento Ágil de software com Arquitetura Multicamadas para Aplicações Java Web**. 2010. 107 f. TCC (Graduação) - Curso de Tecnologia em Banco de Dados, Faculdade de Tecnologia do Estado de São Paulo, São José dos Campos, 2010. Disponível em: <http://fatecsjc.edu.br/trabalhos-de-graduacao/wp-content/uploads/2012/04/BDR2LeiaEdinei.pdf>. Acesso em: 25 out. 2015.

SILVA, Alberto Manuel Rodrigues da; VIDEIRA, Carlos Alberto Escaleira. **UML Metodologias e Ferramentas CASE:** Linguagem de Modelação UML, metodologias e ferramentas CASE na Concepção e Desenvolvimento de Software. Lisboa: Centro Atlântico, 2001. 578 p. Disponível em: <http://www.cesarkallas.net/arquivos/livros/informatica/UML_Metodologias_e_Ferramentas_CASE_portugues_.pdf>. Acesso em: 17 out. 2015.

UML-DIAGRAMS. **UML 2.5 Diagrams**. 2015. Disponível em: <http://www.uml-diagrams.org/uml-25-diagrams.html>. Acesso em: 18 nov. 2015.

VARGAS, Thânia Clair de Souza. **A história de UML e seus diagramas**. Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_721/artigo.tcc.pdf>. Acesso em: 16 out. 2015.

WAZLAWICK, Raul Sidnei. **Análise e Projeto de Sistemas da Informação Orientados a Objeto**. Rio de Janeiro: Campus, 2004. 295 p.