UNIVERSIDADE FEDERAL DO PAMPA

EDUARDO BRUNING

UMA FERRAMENTA DE MODELAGEM COLABORATIVA DE DIAGRAMAS DE CLASSES

Alegrete

2016

EDUARDO BRUNING

UMA FERRAMENTA DE MODELAGEM COLABORATIVA DE DIAGRAMAS DE CLASSES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de *Software* da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de *Software*.

Orientador: João Pablo Silva da Silva

Alegrete

2016

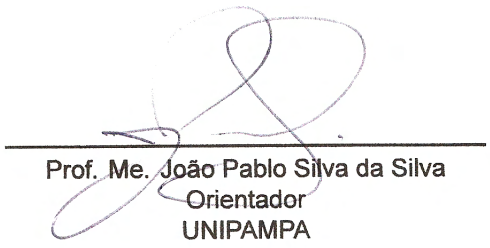
EDUARDO BRUNING

UMA FERRAMENTA DE MODELAGEM COLABORATIVA DE DIAGRAMAS DE CLASSES

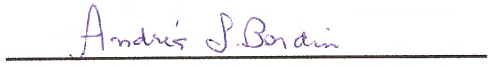
Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de *Software* da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de *Software*.

Trabalho de Conclusão de Curso defendido e aprovado em 02 de Dezembro de 2016.

Banca examinadora:



Prof. Me. João Pablo Silva da Silva
Orientador
UNIPAMPA



Prof. Dra. Andrea Sabedra Bordin
UNIPAMPA

Prof. Dr. Gilleanes Thorwald Araujo Guedes
UNIPAMPA

Este trabalho é dedicado aos meus pais e irmão, pelo exemplo, amor e por sempre estarem ao meu lado nos momentos de dificuldades, tristezas e alegrias.

AGRADECIMENTOS

Agradeço primeiro a Deus, por me manter fortalecido, aos meus pais Cleusa e Senildo, por me apoiarem sempre, dando amor, carinho e incentivo para investir nos estudos. Também agradeço ao meu irmão, Jhosefe por ser um exemplo de dedicação e estar sempre ao meu lado quando preciso de apoio.

Agradeço a todos os professores pelo conhecimento compartilhado durante todo o curso, principalmente ao meu orientador João Pablo, por ter aceito o desafio desse trabalho e pela sua atenção nas questões profissionais e pessoais.

Agradeço aos meus amigos e colegas da faculdade pelo apoio durante todo o caminho, em especial ao Luciano Marchezan, Gustavo da Silva Rodrigues, Kézia Lobo e Douglas Giordano por terem disponibilizado tempo para me auxiliar em questões técnicas específicas durante o trabalho. Também um agradecimento especial aos amigos Vitor Mendonça, Gean Trindade, Diego Vargas, Priscila Perfeito e Adam Prado pelos momentos de apoio, incentivo, brincadeiras, distrações e “puxões de orelha”.

“Não deixem que lhe façam pensar que você não é capaz de fazer algo porque essa pessoa não consegue fazer. Se você deseja alguma coisa, se quer realmente, lute por isso e ponto final.”

(Filme: A procura da Felicidade)

RESUMO

No desenvolvimento de *software* é necessário criar modelos para projetar a construção do sistema, como organizar sua estrutura e como funciona o seu comportamento. Um dos modelos estruturais é o diagrama de classe. Ele é expresso na linguagem padrão de desenvolvimento de *software Unified Modeling Language* (UML), normalmente é construído de forma colaborativa pela equipe de desenvolvimento. Os membros das equipes utilizam diferentes ferramentas para realizar a construção de diagramas de forma colaborativa, mas as ferramentas são inapropriadas e não disponibilizam um bom suporte para o trabalho em conjunto. Com o objetivo de dar suporte para a modelagem colaborativa, este trabalho apresenta uma ferramenta que permite a modelagem de diagramas de classes de forma concorrente pelos membros de uma equipe. Para isso, foi pesquisado o estado da arte dos trabalhos que abordam ferramentas que dão suporte a modelagem colaborativa, buscando tecnologias que auxiliam no desenvolvimento da ferramenta, testado as adequadas ao contexto do trabalho. De acordo com as pesquisas, a ferramenta foi desenvolvida para a plataforma *web* utilizando as tecnologias JavaServer Faces (JSF), uma biblioteca *JavaScript* chamada *GoJS* e o *framework Hibernate*. A verificação foi realizada utilizando testes funcionais e a validação foi feita com possíveis usuários, utilizando algumas métricas oriundas da *Organização Internacional para Padronização* (ISO) 9126. A partir da aplicação dos testes, a ferramenta chegou a uma versão estável e foi possível obter a opinião e avaliação dos usuários. Apesar de alguns erros ocorridos na ferramenta o resultado final foi positivo, levando em consideração a opinião dos usuários que participaram da validação.

Palavras-chave: Modelagem Colaborativa, Diagrama de Classes UML, Produtividade.

ABSTRACT

In software development it is necessary to create models to design the construction of the system, how to organize its structure and how its behavior works. One such model is the class diagram, which is a representation of the software structure. It is expressed in the standard language of software development, the UML, and is usually done collaboratively by the development team. Team members use different tools to construct class diagrams in a collaborative way, but the tools are inappropriate and do not provide good support because they do not have that as their primary goal. With the objective to support collaborative modeling, this paper presents a tool that allows modeling of class diagrams concurrently by members of a team. In order to do this, the state of the art was investigated in the works that deal with tools that support collaborative modeling, searching for technologies that help in the development of the tool and testing those that fit the context of this work. According to the research, the tool was developed for the web platform using these technologies: JSF, a JavaScript library called GoJS and the Hibernate framework. According to the research, the tool was developed for the web platform using JSF technologies, a JavaScript library called GoJS and the Hibernate framework. The verification was performed using manual functional tests and validation was done with possible users, using some metrics from ISO 9126. From the application of the tests, the tool arrived at a stable version and it was possible to obtain the opinion and evaluation of the users. Although some errors occurred in the tool the final result was good, taking into account the opinion of the users who participated in the validation.

Key-words: Collaborative modeling, Class Diagrams UML, Productivity.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AJAX *Asynchronous Javascript and Extensible Markup Language (XML)*

API Application Programming Interface

BPMN *Business Process Model and Notation*

CASE *Computer-Aided Software Engineering*

coBPM *collaborative Business Process Modeler*

CRC *Class-responsibility-collaboration*

DSML *Domain-Specific Modelling Languages*

ISO *Organização Internacional para Padronização*

JSF JavaServer Faces

JSON *JavaScript Object Notation*

LGPL *GNU Lesser General Public License*

MDE *Model Driven Engineering*

OMT *Object Modeling Technique*

OOSE *Object-Oriented Software Engineering*

POJO *Plain Old Java Objects*

SDK *Software Development Kit*

SQL *Structured Query Language*

UML *Unified Modeling Language*

XML *Extensible Markup Language*

XMPP *Extensible Messaging and Presence Protocol*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de *software* é composto de diferentes etapas, a modelagem está presente em grande parte delas. Ela pode ser comparada com a construção de plantas de casas, que tem por objetivo representar a sua estrutura. Com um *software*, a modelagem compartilha o mesmo objetivo, descrever como deve ser a estrutura do sistema e também o seu comportamento. Booch, Rumbaugh e Jacobson (2006) afirmam que a modelagem de *software* é a atividade de construção de modelos que descrevem as características e o comportamento do sistema. Bezerra (2007) conceitua modelos como uma representação idealizada de um sistema a ser construído, já Booch, Rumbaugh e Jacobson (2006) têm uma visão mais abrangente, afirmando que modelos são uma simplificação da realidade.

Os modelos de *software* são criados com o objetivo compreender o problema, visualizar como o sistema é, ou como ele é desejado, definir sua estrutura e como deve acontecer o seu comportamento (BOOCH; RUMBAUGH; JACOBSON, 2006). Para construir esses modelos a equipe de desenvolvimento precisa trabalhar em conjunto, para isso podem ser utilizadas diferentes estratégias, como quadros brancos, dispositivos móveis e ferramentas de apoio, como as ferramentas CASE (LARMAN, 2007; BARTELT; VOGEL; WARNECKE, 2013). As equipes que trabalham à distância recorrem para *softwares* de compartilhamento de tela e arquivos, ferramentas de desenho, sistemas de conferência, *e-mails* e até mesmo redes sociais com o objetivo de trabalhar em conjunto com os demais membros da equipe (BARTELT; VOGEL; WARNECKE, 2013; OSMAN, 2013).

Como os modelos podem ser feitos por diferentes pessoas e ferramentas, é preciso uma linguagem que expresse as informações do modelo, para ser possível realizar a troca de dados sem problemas de compatibilidade. A UML é a linguagem padrão para todas as etapas de modelagem de *software*. Ela surgiu de três modelos propostos na década de 90, o modelo Booch, *Object-Oriented Software Engineering* (OOSE) e o *Object Modeling Technique* (OMT). Cada modelo possuía vantagem em uma etapa do ciclo de vida do *software*. Consequentemente, motivou a criação de uma linguagem unificada, a UML (BOOCH; RUMBAUGH; JACOBSON, 2000). Sommerville (2007) afirma que após a unificação dos modelos propostos por Booch, Rumbaugh e Jacobson, a UML se tornou padrão para a modelagem de *software*. Como ela dispõe de diferentes modelos para a especificação de *software*, utilizá-los interligando um modelo ao outro, ajuda a obter uma visão mais detalhada do sistema.

1.1 Motivação

Quando os membros de uma equipe trabalham em conjunto dentro da mesma sala, Larman (2007) relata a utilização de quadros brancos e Osman (2013) apresenta o uso de cartões *Class-responsibility-collaboration* (CRC), canetas e papéis. Deste modo, Osman (2013) afirma que a colaboração não é um problema, mas para documentar as informações dos modelos criados, se torna demorado e trabalhoso, pois é necessário transcrever o modelo para uma ferramenta. Osman (2013) também fala do uso da UML nestes casos, porém, um membro por vez pode modelar, enquanto os demais auxiliam e discutem em conjunto com a equipe.

Além das equipes que trabalham em sessões de modelagem na mesma sala, o autor Osman (2013), apresenta um relato das dificuldades existentes para os membros das equipes que trabalham distantes geograficamente. Osman (2013) afirma que nesses casos as sessões envolvem canais de comunicação, como *e-mail* e *Skype*[1] e meios de compartilhamento de arquivos como *Dropbox* [2], *Google Drive* [3] e anexos de *e-mails*. Osman (2013) friza que esses modos afetam negativamente a produtividade da equipe, pela quebra de fluxo na colaboração, além de cada membro não ser capaz de ver as mudanças que os demais realizaram até que ele compartilhe. Além disso, eventualmente é necessário realizar a sincronização dos modelos para resolver conflitos de duplicidade de arquivos.

Os problemas na modelagem colaborativa envolvem vários tipos de modelos, um deles é o diagrama de classes, este modelo além de ser utilizado na fase de projeto, também pode ser usado nas etapas de análise para o entendimento do problema. Isso o torna bastante utilizado e importante para o desenvolvimento de *software*. Dada a importância do diagrama de classes para o desenvolvimento, é interessante uma ferramenta que de suporte a modelagem colaborativa para diagramas de classes. De acordo com nossa revisão sistemática, abordada no Capítulo 3, existem poucas ferramentas disponíveis para possibilitar que os membros da equipe de desenvolvimento possam trabalhar de forma concorrente em um mesmo diagrama de classes. Algumas, ao tentar abranger a maior quantidade de especificações em diversas linguagens, não consegue dar um suporte adequado a UML, que define a especificação do diagrama de classes.

1.2 Objetivos

Nosso trabalho tem como objetivo geral o desenvolvimento de uma ferramenta de suporte a modelagem colaborativa de diagramas de classes da UML. Para pos-

[1] *Skype* <https://www.skype.com>
[2] *Dropbox* <https://www.dropbox.com>.
[3] *Google Drive* <http://drive.google.com/>.

sibilitar que a equipe de desenvolvimento possa criar modelos de classes de forma concorrente reduzindo os problemas que abordamos na Seção 1.1.

Para atingir o propósito desse trabalho de conclusão de curso, é necessário alcançar os seguintes objetivos específicos:

- modelar o diagrama de classes através de uma interface *web*;
- permitir que várias pessoas modelem o mesmo diagrama concorrentemente;
- proporcionar um aumento de produtividade por meio da utilização da ferramenta.

1.3 Metodologia de trabalho

Nas primeiras etapas de desenvolvimento do trabalho investigamos o estado da arte, buscando por ferramentas de modelagem colaborativa. Utilizamos um mapeamento sistemático para buscar trabalhos relacionados ao nosso contexto. De acordo com essas primeiras etapas, obtivemos um panorama sobre os trabalhos desenvolvidos e definimos o escopo do nosso trabalho.

Estudamos os conceitos e técnicas da linguagem UML, ferramentas CASE, as formas de modelagem de *software*, diferentes arquiteturas de *software* e tecnologias para desenvolvimento na plataforma *web*. Definimos e testamos as tecnologias para utilizar na ferramenta e desenvolvemos uma primeira versão, contemplando grande parte do contexto previsto. Na sequência, realizamos iterações de testes funcionais manuais, para isso criamos os casos de testes e dois usuários executavam eles na ferramenta, os erros encontrados eram refatorados e consequentemente eram refeitos os testes. Este processo com dois usuários se aplicou até a terceira iteração, nas duas últimas, apenas um usuário realizou os testes, chegando a uma versão estável.

Por fim, realizamos um teste com possíveis usuários finais para validar a ferramenta, para avaliá-la utilizamos a 9126-1 (2003), criamos um questionário com métricas que se adequavam ao nosso contexto e os usuários avaliaram a ferramenta, de acordo com a experiência que tiveram.

1.4 Organização do restante do trabalho

A sequência do nosso trabalho está organizado e estruturado da seguinte maneira:

- No **Capítulo 2** apresentamos as ferramentas CASE, a linguagem UML, descrevemos a modelagem colaborativa de *software* e as tecnologias *web*.

- No **Capítulo 3** retratamos a metodologia utilizada para coleta dos trabalhos relacionados com essa pesquisa e abordamos seus os resultados obtidos.
- No **Capítulo 4** exibimos a ferramenta de modelagem colaborativa, mostrando alguns artefatos do seu desenvolvimento.
- No **Capítulo 5** mostramos as técnicas e artefatos gerados nos testes e na validação.
- No **Capítulo 6** apontamos as conclusões dessa pesquisa, relatamos as contribuições e discutimos sobre trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA E TECNOLÓGICA

Neste capítulo apresentamos os conceitos fundamentais necessários para o entendimento do trabalho. Na Seção 2.1 abordamos a visão da linguagem UML. Na Seção 2.2 apresentamos o que são e para que servem as ferramentas CASE. Na Seção 2.3 descrevemos o que é modelagem de *software* e abordamos alguns aspectos da modelagem colaborativa. Na Seção 2.4 abordamos dois padrões de projetos utilizados para o desenvolvimento de *software*. Na Seção 2.5 apresentamos algumas tecnologias para *web* que são utilizadas no trabalho. Na Seção 2.6 descrevemos o *framework Hibernate* utilizado para facilitar o desenvolvimento da ferramenta. Por ultimo na Seção 2.7 encerramos com as lições do capítulo.

2.1 Linguagem UML

A UML é uma linguagem de modelagem, que possui um vocabulário e regras com seu foco voltado para representação conceitual e física de um *software* (BOOCH; RUMBAUGH; JACOBSON, 2006). Larman (2007) complementa afirmando que é uma linguagem visual para a especificação, construção e documentação de artefatos de *software*. Cardoso (2003) afirma que ela surgiu com a ideia de reunir os melhores modelos propostos na época em apenas um modelo. Desta forma Bezerra (2007) relata que a UML se tornou a linguagem padrão no desenvolvimento de sistemas. Sua definição está em constante melhoria, várias atualizações foram realizadas para deixá-la mais clara e útil. Atualmente ela está na versão 2.5 (OMG, 2015).

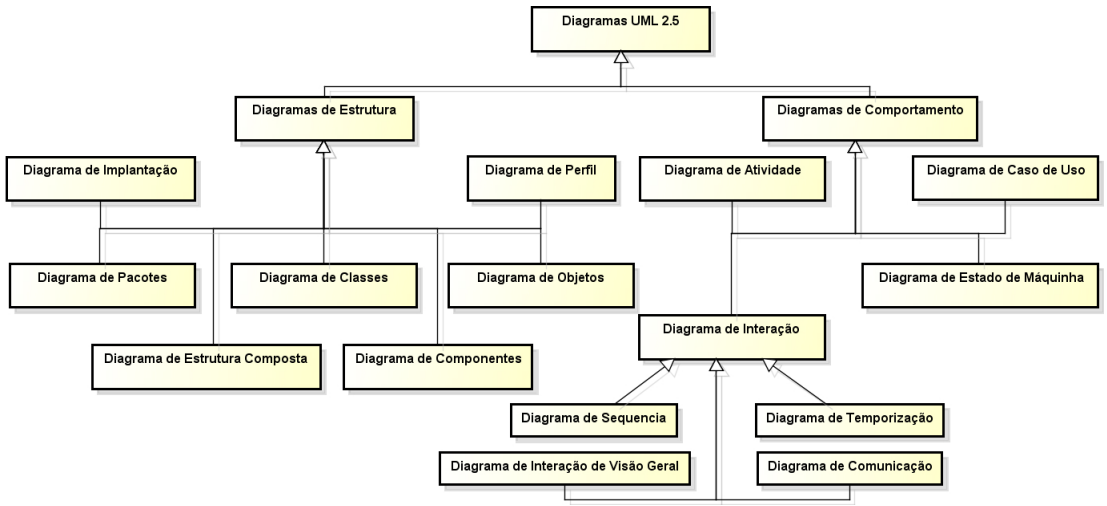
Bezerra (2007) afirma que cada elemento gráfico da UML possui sintaxe e semântica, ou seja, cada elemento corresponde a uma determinada forma e possui um significado, definindo para qual objetivo deve ser usado. Booch, Rumbaugh e Jacobson (2006) dizem que a UML não é uma linguagem visual de programação, porém seus modelos podem ser utilizados para modelar diversos sistemas em diferentes linguagens, até mesmo uma representação do banco de dados.

2.1.1 Diagramas

Segundo Bezerra (2007), diagramas são uma apresentação de coleções de elementos gráficos que tem um significado predefinido. Atualmente na versão 2.5 da UML a OMG (2015) separa em dois grandes grupos, os diagramas estruturais e os comportamentais, como exibe a Figura 1. A OMG (2015) define que os estruturais demonstram a estrutura estática do sistema e como é a relação entre os níveis do sistema. Já os comportamentais demonstram o comportamento dinâmico dos obje-

tos no sistema. A grande vantagem apresentada por Bezerra (2007) na utilização de diversos diagramas, é a possibilidade de construir o *software* através de várias perspectivas, melhorando sua qualidade e garantindo o entendimento do sistema em desenvolvimento.

Figura 1 – Diagramas da UML 2.5.



Fonte: Adaptado de: OMG (2015)

A diferença de diagramas da UML 2.0 para 2.5 é de apenas um, o diagrama de perfil, adicionado aos diagramas estruturais. Os demais são basicamente os mesmos, com algumas melhorias em relação a versão anterior.

2.1.1.1 Diagrama de classes

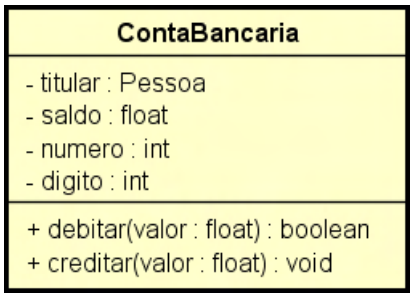
Entre os diagramas estruturais da UML está o diagrama de classes, que tem como objetivo descrever as classes que compõem um sistema com suas características, operações e as relações entre as outras classes que formam o sistema.

Na Figura 2 é mostrado um exemplo de classe da UML, ela é representada visualmente como uma caixa. A classe segundo a OMG (2015), é um elemento abstrato que representa um conjunto de objetos com seus atributos e métodos.

Tratando da sua estrutura gráfica, podemos ver na Figura 2 que no compartimento superior fica localizado o nome da classe em negrito. A baixo do nome da classe, são exibidos os seus atributos e no compartimento inferior os métodos. Ambos são alinhados a esquerda e com sua primeira letra em minúsculo.

Como é possível ver na Figura 2, antes do nome dos atributos e métodos existe uma notação, estas notações representam a visibilidade daquele elemento. As visibilidades podem ser as seguintes:

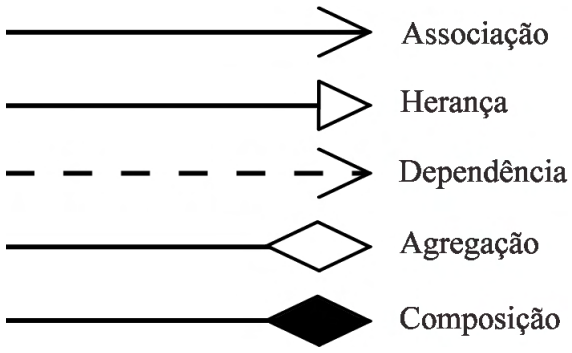
Figura 2 – Classe da UML.



- + pública: outras classes podem ter acesso ao elemento;
- - privada: acessado somente direto pela própria classe;
- # protegida: acessado pela classe e suas subclasses;
- ˜pacote: acessado por todos dentro do pacote.

Tratando dos relacionamentos, no diagrama de classes eles descrevem como as classes interagem umas com as outras e podem também definir responsabilidades (OMG, 2015). Os tipos de relacionamentos possíveis são: dependência, associação, agregação, composição e herança. Na Figura 3 exibimos como as relações são graficamente.

Figura 3 – Relações do diagrama de classes da UML.



A relação de associação descreve um vínculo entre duas classes, ou seja, a instância de uma classe está de alguma forma ligada a instância da outra classe. A herança ou generalização define uma relação onde existe uma super-classe e uma sub-classe, sendo que a sub-classe herda os atributos e métodos da super-classe. A relação de dependência indica uma relação fraca entre dois objetos de duas classes, denota uma conexão entre elementos do modelo dependentes e independentes. A agregação define uma relação de todo-parte, pois demonstra que informações de um

objeto de uma classe necessitam ser complementadas com outra classe. Já na composição, a relação define um vinculo mais forte de objeto-todo e objeto-parte, onde o objeto parte não existe ou não faz sentido sem o todo, mas o todo existe sem o parte (OMG, 2015).

2.2 Ferramentas CASE

Para o desenvolvimento de *software* são utilizadas ferramentas para auxiliar o desenvolvedor e aumentar sua produtividade, essas ferramentas são chamadas de CASE. Ferramentas CASE são uma classificação de todas as ferramentas baseadas em computador que auxiliam as atividades de engenharia de *software*, da análise até os testes, ou seja, são ferramentas automatizadas que tem como objetivo dar suporte ao desenvolvedor durante todas as etapas do ciclo de desenvolvimento de *software* (SCHACH, 2009; SOMMERVILLE, 2007).

Existem diferentes formas para classificar as ferramentas CASE, abordamos neste trabalho a classificação na perspectiva dos autores Sommerville (2007) e Schach (2009). A classificação é dividida em três categorias: ferramenta, plataforma (bancada de trabalho) e ambiente, a Figura 4 demonstra um exemplo para esta classificação.

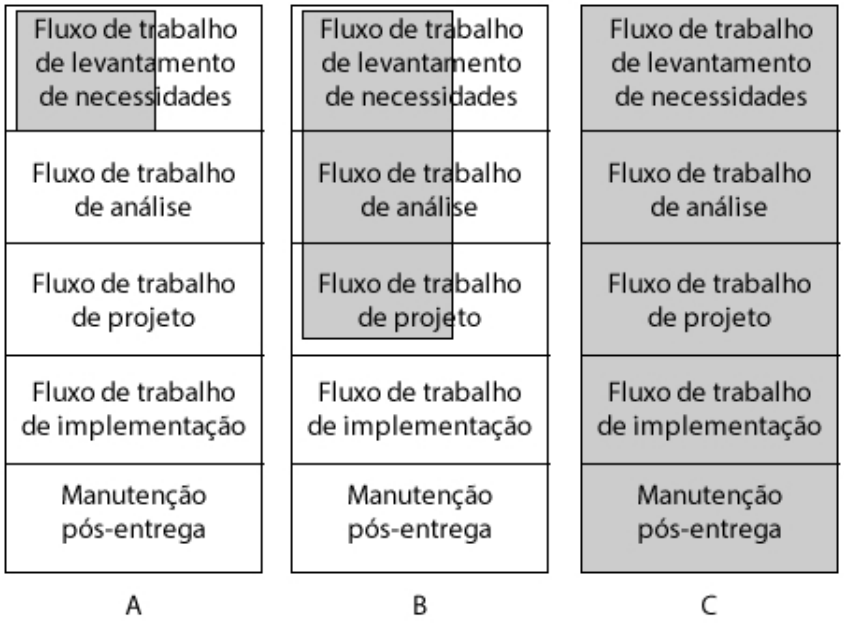
Na categoria de **ferramenta**, Schach (2009) afirma que ela é a forma mais simples de CASE, sendo um produto que auxilia em apenas um aspecto da produção de *software*. Sommerville (2007) relata que elas apoiam atividades individuais de processo, como verificação de consistência do projeto. Outros exemplos são *softwares* que auxiliam com representações gráficas do projeto, como fluxogramas e diagramas UML.

Na categoria **plataforma**, CASE é um conjunto de ferramentas que unidas suportam uma ou duas atividades, sendo essas, um conjunto de tarefas relacionadas, por exemplo, a codificação, que inclui as atividade de edição, compilação, testes, depuração, entre outras (SCHACH, 2009; SOMMERVILLE, 2007).

Na categoria **ambiente**, uma CASE suporta todo o processo, ou pelo menos grande parte dele, normalmente eles são formados por diversas plataformas interligadas (SCHACH, 2009; SOMMERVILLE, 2007).

Essa forma de classificação ajuda bastante no entendimento de CASE, é possível visualizar na Figura 4 que várias ferramentas formam uma plataforma e que o ambiente pode ser formado por plataformas, mas a característica notória do ambiente é apresentar um suporte maior, ou total, ao processo de desenvolvimento.

Figura 4 – Representação de (A) ferramenta, (B) plataforma e (C) ambiente.



Fonte: Adaptado de Schach (2009, p. 131)

2.3 Modelagem colaborativa

A modelagem colaborativa é um dos tipos de modelagem de *software*, que por sua vez é definida como:

> A modelagem de sistemas de *software* consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se várias perspectivas diferentes e complementares (BEZERRA, 2007).

A modelagem colaborativa mais conhecida e utilizada é descrita por Larman (2007). Ela consiste em membros de uma equipe reunidos em uma sala discutindo e rabiscando diagramas em quadros. Complementarmente, Larman (2007) apresenta três formas para realizar essa modelagem: divide os membros em pares, os quais trabalham isolados; diversos pares no mesmo ambiente com um mediador; ou ainda a equipe inteira trabalhando junto, modelando o *software* em diferentes perspectivas.

Bartelt, Vogel e Warnecke (2013) apresentam uma forma de modelagem na mesma perspectiva que Larman (2007). Os membros de uma equipe se reúnem no mesmo local e utilizam dispositivos móveis com telas grandes e sensíveis ao toque para a equipe realizar a modelagem. Dirix (2013) nomeia esse tipo de colaboração como “natural” porque é possível facilmente encontrar uma resposta das dúvidas do modelo com os membros da equipe.

Diferente da colaboração natural, equipes que trabalham geograficamente distantes têm mais dificuldade em obter respostas imediatas sobre os modelos. Para tratar essas questões são utilizadas algumas estratégias como a descrita por Bartelt, Vogel e Warnecke (2013), na qual são usados *softwares* de compartilhamento de tela em conjunto com *softwares* de desenho e, se necessário, um sistema de conferência telefônica para que a equipe possa dialogar durante a construção do modelo.

Além dessa estratégia, Osman (2013) apresenta outra em que os membros das equipes utilizam redes sociais, e-mails e mecanismos de compartilhamento de arquivos para estabelecer a comunicação. Nesse caso é necessário realizar uma divisão prévia do trabalho para posteriormente integrar os modelos, pois cada membro se comunica com os demais, mas eles não possuem acesso para visualizar as alterações em tempo real.

2.4 Padrões de projetos

Padrões de projetos segundo Larman (2007), são soluções testadas para determinados problemas. Um padrão de projeto descreve uma solução genérica, utilizada para resolver um problema específico que pode ser adaptada de acordo com a necessidade da solução. Gamma (2009) afirma que os padrões são soluções eficientes, comprovadas e amplamente utilizadas para resolver problemas comuns no desenvolvimento de softwares. As soluções se tornam padrão porque foram usadas em diferentes projetos e por ter seu funcionamento comprovado.

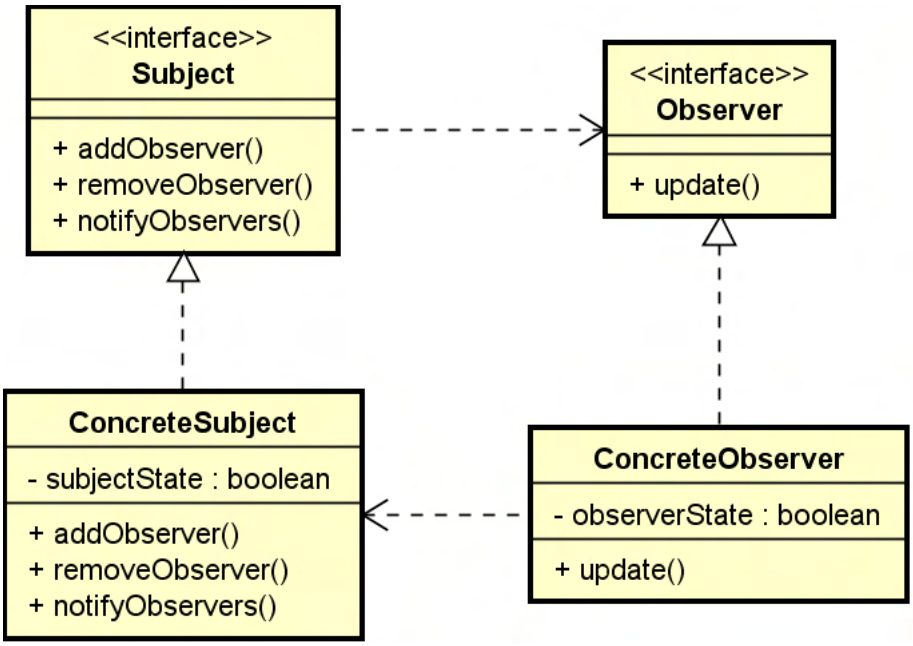
2.4.1 Padrão *observer*

O padrão de projeto *Observer* pode ser explicado como assinaturas de jornais, existe uma editora que é responsável por publicar as edições e algumas pessoas que assinam para receber as edições do jornal. A cada nova edição lançada pelo jornal, os assinantes as recebem. Se o assinante não deseja mais receber o jornal, ele pode cancelar sua assinatura. Podem surgir também novos assinantes para receber as edições que o jornal publica (LARMAN, 2007).

O padrão *Observer* tem essa mesma funcionalidade, A editora é o *Subject* e os assinantes são o *Observer*. Os *Observers* se registam no *Subject* e passam a receber as atualizações dos dados do *Subject*. Quando não for mais pertinente receber essas atualizações o *Observer* pode cancelar o seu registro, desta forma não receberá mais as informações (GAMMA, 2009). Na Figura 5 é apresentado o diagrama de classes do padrão *Observer*.

A *ConcreteSubject* define o comportamento dos métodos *addObserver*, para adicionar um *Observer* e *removeObserver*, para remover um *Observer*. O método

Figura 5 – Diagrama de classes para demonstrar o funcionamento do *Observer*.



Adaptado de Medeiros (2016)

notify é usado para atualizar todos os observadores registrados sempre que o estado do Subject mudar. Já o *ConcreteObserver* implementa a interface *Observer* e tem o método *update*, que é chamado quando o estado do *Subject* for alterado.

2.4.2 Padrão *singleton*

O padrão de projeto *Singleton* tem como objetivo, garantir a existência de apenas uma instância de determinada classe, com visibilidade e acessibilidade global em um determinado escopo de projeto. Alguns projetos às vezes precisam que algumas classes tenham apenas uma instância. Suponha, um sistema onde todo o usuário que entrar precisar registrar seu acesso. Cada objeto usuário informa o objeto que utiliza o padrão *Singleton* que acessou o sistema. Apenas esse objeto é responsável por registrar essas informações. No exemplo da Código 2.1 é possível visualizar a implementação do padrão na linguagem Java, onde existe um atributo do tipo da classe e um método que retorna a instância da classe. O método verifica se já existe uma instância criada. Se existe, a retorna, caso não, o método cria e retorna a instância. Desta forma é assegurado que só terá um instância dessa classe (GAMMA, 2009).

Código 2.1 – Exemplo de implementação do padrão *Singleton*.

```
public class Repositorio{
  private static Repositorio uniqueInstance = new Repositorio();
```

```
   public Repositorio(){
   }

   /**
   * implementacao do padrao Singleton
   *
   * @return o Repositorio
   */
   public static Repositorio getInstance(){
      if(uniqueInstance == null){
         uniqueInstance = new Repositorio();
      }
      return uniqueInstance;
   }
}
```

2.5 Tecnologias para *Web*

2.5.1 *JavaServer Faces*

A tecnologia *JavaServer Faces* é um *framework* de componentes de interface do lado servidor para aplicações *web* baseados na tecnologia Java. Foi desenvolvida pela comunidade *Java Community Proccess*, estabelecendo o padrão para a construção de interfaces com o usuário do lado servidor. A Oracle (2015) define os seguintes componentes como os principais do JSF:

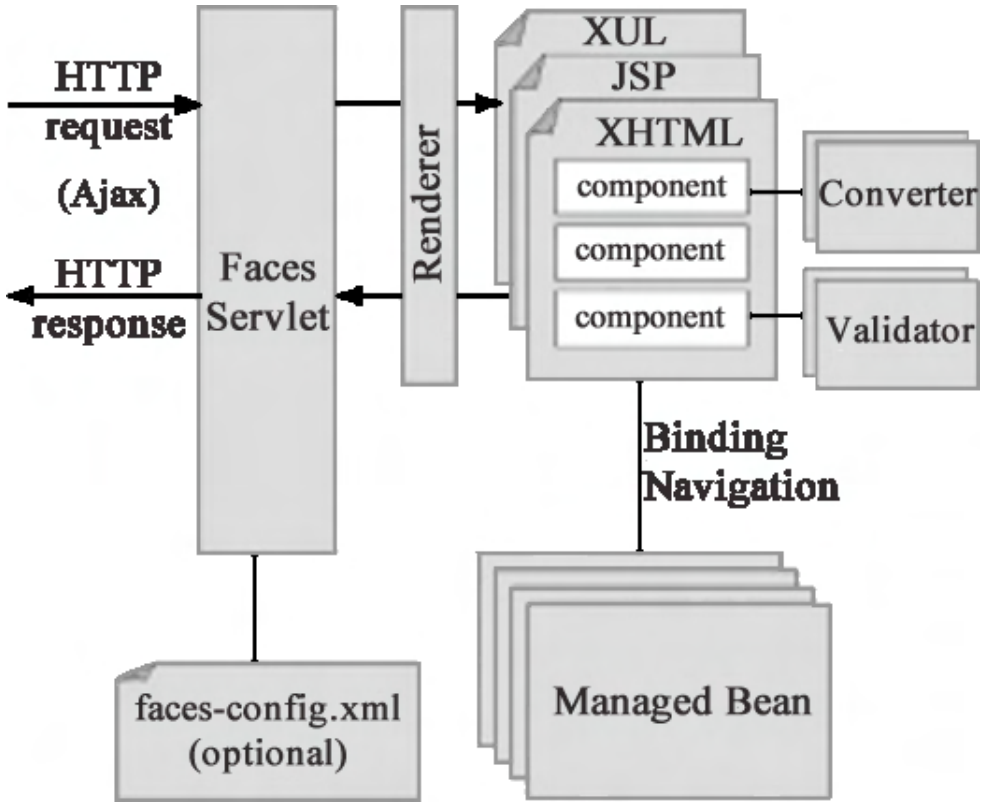
- Uma Application Programming Interface (API) para representar os componentes de interface do usuário e o controle do seu estado. Fornece extensibilidade para algumas outras características como: manipulação dos eventos, validação do lado servidor, definição de navegabilidade entre páginas e suporte para internacionalização e acessibilidade (ORACLE, 2015).
- Bibliotecas de *tags* para a adição de componentes para páginas *web* e para conectar componentes de objetos do lado servidor (ORACLE, 2015).

A tecnologia JSF fornece uma forma de programação bem definida e várias bibliotecas contendo manipuladores que implementam as *tags* de componentes. Esses recursos tem como objetivo facilitar a construção e manutenção das aplicações *web* com interfaces de usuários do lado servidor.

A Figura 6 ilustra uma requisição em uma aplicação JSF, feita através do navegador. Ao ser processada, constrói a página com seus componentes. Sua nave-

gação ocorre através de um *Managed Bean*, que define propriedades e funções para os componentes de interface do usuário da página. Podemos perceber a utilização de um arquivo opcional, o *faces-config.xml*, usado para definir regras de navegação das páginas (ORACLE, 2015).

Figura 6 – Requisição em JSF.



Adaptado de Oracle (2015)

2.5.2 Biblioteca GoJS

GoJS é um biblioteca *JavaScript*, rica em recursos para a implementação de diagramas, tabelas e gráficos complexos e interativos nos diferentes navegadores e plataformas, sem a necessidade da utilização de *plug-ins*. *GoJS* permite implementar vários recursos avançados para a interatividade do usuário como mover, deletar, editar, adicionar nodos e criar relações entre os nodos. Também disponibiliza *layouts*, modelos e manipuladores de eventos (NORTHWOODS, 2016). Seus códigos estão disponíveis em <https://github.com/NorthwoodsSoftware/GoJS> e ela é mantida pela empresa *Northwoods Software Corporation* (NORTHWOODS, 2016).

2.5.2.1 Conceitos *GoJS*

O diagrama é formado por nodos que podem ser conectados por links, organizados segundo um *layout*. O diagrama tem um modelo que interpreta os dados para determinar as relações entres os nodos e dos nodos com os grupos. Da mesma forma, os nodos e os *links* normalmente possuem um modelo definido, chamado de painel, para declarar sua aparência e comportamento. Os nodos podem ter textos, imagens ou formas. Existem alguns modelos padrões para todas as partes, que podem ser personalizados de acordo com o desejo do desenvolvedor (NORTHWOODS, 2016).

Os nodos podem ser posicionados na tela manualmente ou podem ser dispostos automaticamente através do seu *layout*. Os diagramas possuem uma vasta gama de ferramentas para executar tarefas, como a seleção de nodos, movimentação, novas relações entre nodos e o tratamento de eventos do teclado. É possível alterar a aparência do modelo ao selecionar um nodo ou *link* e também permitir que eles sejam redimensionados. Desta forma, dentro do diagrama estão todas as camadas, o que inclui os nodos e *links* que são compostos por painéis, podendo conter texto, formas e imagens. Toda essa hierarquia de objetos, constitui na memória uma árvore visual de tudo que pode ser elaborado pelo diagrama. O GoJS é uma biblioteca simples, poderosa e bem documentada (NORTHWOODS, 2016).

2.6 *Framework Hibernate*

Hibernate é um *framework* Java distribuído sob a licença *GNU Lesser General Public License* (LGPL). O *Hibernate* é *framework* para mapeamento objeto-relacional, por meio de arquivos XML ou anotações Java. Ele tem como finalidade simplificar o mapeamento dos atributos entre uma base de dados relacionais e um modelo orientado a objeto, em especial, no desenvolvimento de consultas e atualização de dados (HAT, 2016).

O *Hibernate* possibilita desenvolver classes de persistência usando o Java convencional, com associação, composição, herança, polimorfismo e as coleções Java. Transforma as classes Java em tabelas do banco de dados relacional e os tipos do java nos tipos do *Structured Query Language* (SQL)

Na Figura 7 exibimos a arquitetura do *Hibernate* em alto nível. A aplicação persiste os objetivos Java comuns chamados de *Plain Old Java Objects* (POJO). Sob responsabilidade do *Hibernate* está os mapeamentos XML, responsáveis para descrever os campos, associações e subclasses e o arquivo de propriedades, define as configurações do banco de dados para estabelecer a conexão. Sendo assim, podemos ver pela Figura 7 que o *Hibernate* faz a comunicação da aplicação com o banco de dados, encapsulando as operações e facilitando para o desenvolvedor.

Figura 7 – Arquitetura do *Hibernate*

Adaptado de Hat (2016)

2.7 Lições do capítulo

No decorrer deste capítulo apresentamos diversos conceitos e tecnologias, essas informações são importantes para entender o desenvolvimento do trabalho. Os conceitos mais teóricos que apresentamos como a UML, ferramentas CASE e a modelagem colaborativa, servem como base para mostrar o objetivo da nossa ferramenta.

Para desenvolver a ferramenta necessitamos utilizar as tecnologias disponíveis. Pesquisamos as que se adequam ao nosso contexto e detalhamos como funcionam e para que servem. Sendo elas o JSF, a biblioteca *JavaScript GoJS* e o *framework Hibernate*. Além das tecnologias apresentadas, exibimos dois padrões de projeto que nos ajudam a solucionar o principal objetivo do nosso trabalho, que é a colaboração.

3 TRABALHOS RELACIONADOS

Neste capítulo apresentamos os trabalhos obtidos com a execução de um mapeamento sistemático. Na Seção 3.1 apresentamos como foi realizado o mapeamento sistemático, na Seção 3.2 são apresentados os resultados de acordo com as questões de pesquisas e na Seção 3.3 são apresentadas as considerações finais dos trabalhos relacionados.

3.1 Protocolo do mapeamento

O objetivo deste mapeamento foi investigar ferramentas de modelagem colaborativa de diagramas UML. Nesse sentido, definimos as seguintes questões de pesquisa.

- Quais são as plataformas onde as atuais soluções rodam?
- Quais são as tecnologias que as soluções utilizam?
- Que estratégias são usadas para tratar a edição colaborativa?

O mecanismo de busca escolhido foi o Google Scholar [1] por indexar a maior parte das bases, tornando a pesquisa mais completa. Utilizamos como palavras-chaves “UML AND tool AND ‘collaborative modeling’ ”, considerando apenas publicações em inglês. Os critérios para a seleção dos trabalhos foram divididos em critérios de inclusão e exclusão. Nos critérios de inclusão decidimos filtrar pelo ano das publicações, buscando trabalhos recentes com seu conteúdo vinculado a esta pesquisa, como a seguir listado.

- Artigos completos publicados em eventos e periódicos nos anos de 2011 a 2015;
- Artigos que abordam questões sobre o desenvolvimento de ferramentas de edição colaborativa;
- Trabalhos que apresentem evidências que sustentem as conclusões apresentadas.

Para os critérios de exclusão, descartamos os trabalhos curtos e superficiais que não apresentam informações detalhadas da sua realização, desta forma os critérios são:

[1] Google Scholar <http://scholar.google.com/>.

- Artigos do tipo: revisões sistemáticas, mapeamento sistemático, relatórios técnicos ou *survey*;
- Artigos repetidos, versões curtas ou com menos de 6 páginas.

Os resultados obtidos foram classificados de acordo com as questões de pesquisa, separando-os pelas plataformas, tecnologias e estratégias de colaboração. Na última etapa, decidimos extrair dos trabalhos, os seus objetivos, a metodologia utilizada e os resultados obtidos, para que pudéssemos ter um panorama geral da pesquisa realizada e avaliar se ela é útil para este trabalho.

3.2 Resultado

A aplicação do processo de busca resultou em 620 trabalhos, com a aplicação dos critérios de inclusão, reduziu-se para 296 e após a execução dos critérios de exclusão resultou em 28. Os trabalhos foram estudados para analisar se respondem as questões de pesquisa, deste modo, chegando a 11 trabalhos que contemplam as questões definidas.

Na Tabela 1 exibimos um resumo das ferramentas apresentadas nos trabalhos selecionados. Identificamos para cada ferramenta o tipo de plataforma que ela foi desenvolvida, o tipo de comunicação escolhida e quais tecnologias utilizadas para construí-la. Notamos que maior parte das ferramentas foram desenvolvidas para a plataforma *web* e o tipo de comunicação mais utilizado foi a síncrona. Percebemos que em relação as tecnologias, ocorre uma variação bem grande, apenas o *JavaScript(JS)* e o Eclipse que são utilizados em duas ferramentas diferentes.

É importante salientar que alguns trabalhos não respondem as três questões que definimos, sendo assim, na Tabela 1 utilizamos o sinal "-" para as questões não respondidas e para as respondidas a letra "**X**".

Tabela 1 – Resumo das ferramentas encontradas de acordo com as questões de pesquisa.

Ferramentas	Plataforma			Comunicação		Tecnologias utilizadas nas soluções								
	web	*desktop*	móvel	assín.	sínc.	*JS*	*AJAX*	*HTML5*	*Widget*	Eclipse	*Hibernate*	*Java*	*Telex*	*cometD*
SyncLD	X				X				X					
Sawa	X				X	X		X						
WebGME	X			X			X							
HawEye	X			-	-		-	-	-	-	-	-	-	-
Collaboro		X		-	-					X				
CoMoMod		X			X	-	-	-	-	-	-	-	-	-
CoBPM			X		X	-	-	-	-	-	-	-	-	-
C-Praxis	-	-	-	-	-								X	
Lins et al. (2011)		X		X						X				
Azevedo et al. (2013)	X			-	-	X					X	X		X
Jiang, Zhang e Zhao (2014)	X			X		-	-	-	-	-	-	-	-	-

3.2.1 Quais são as plataformas onde as atuais soluções rodam?

As plataformas onde as ferramentas de modelagem são projetadas são: *Web*, *Desktop* e móvel. Para a plataforma *Web* os autores Nicolaescu, Derntl e Klamma (2013) apresentam uma ferramenta chamada *SyncLD* que apoia a edição colaborativa de modelos de design, ela trabalha com colaboração síncrona. Já Jiang, Zhang e Zhao (2014) desenvolveram uma com a abordagem colaborativa voltada para a modelagem conceitual. Ela disponibiliza a edição dos modelos de forma indireta, onde os modeladores editam em uma área individual e a ferramenta mescla em uma área colaborativa. Na área de modelagem UML, o autor Osman (2013) desenvolveu uma ferramenta chamada de *Sawa*, para realizar a modelagem colaborativa de diagramas UML em tempo real.

Em um contexto diferente, Azevedo et al. (2013) desenvolveram um editor colaborativo de diagramas de *brainstorming*, com foco em auxilar os ambientes de aprendizagem no ganho de desempenho dos usuários. Para a modelagem de domínios específicos, a aplicação *WebGME* apresentada por Maróti et al. (2014), utiliza uma infraestrutura em nuvem e possui como objetivo, auxiliar na modelagem colaborativa de projeto de linguagem específica de domínio.

Tratando-se de ferramentas *Web* focadas na modelagem de negócios, Cognini et al. (2013) apresentam a *HawkEye*, para a modelagem de negócios inter-organizacionais de instituições públicas. Para plataformas móveis, Döweling et al. (2013) desenvolveram a *collaborative Business Process Modeler* (coBPM) focada na modelagem colaborativa de processos de negócios. No mesmo contexto, porém para plataforma *desktop*, os autores Dollmann et al. (2011) descrevem sua ferramenta chamada de CoMoMod para a modelagem de processos de negócios de forma colaborativa. Bem como Lins et al. (2011) que desenvolveram um ambiente colaborativo para modelar processos de negócios. Em um contexto diferente, outra ferramenta *Desktop* é utilizada por Izquierdo et al. (2013), estendendo-a para promover a participação mais ativa dos utilizadores finais no processo de *Domain-Specific Modelling Languages* (DSML).

3.2.2 Quais são as tecnologias que as soluções utilizam?

As tecnologias utilizadas pelas aplicações são variadas, como a dos autores Nicolaescu, Derntl e Klamma (2013) que desenvolveram a ferramenta *SyncLD* baseada em *ROLE Software Development Kit* (SDK), que é a plataforma de desenvolvimento de *Widgets* na linguagem Java. O SDK fornece todos os meios técnicos para a prestação e gestão de *Widgets* na *Web* usando o *OpenSocial* um contêiner padrão do *Apache Shindig*. Já Michaux et al. (2011) descrevem a C-Praxis, para modelagem colaborativa e para auxiliar na *Model Driven Engineering* (MDE), ela é baseada em Telex, uma camada de *middleware* projetada para aplicações de *softwares* colabora-

tivos. Telex além de detectar conflitos entre operações simultâneas trata a replicação, armazenamento, controle de acesso e consistência dos modelos.

A aplicação *Sawa* de Osman (2013) é desenvolvida em HTML5 utilizando *JavaScript* como linguagem de *script* por ter um grande apoio da comunidade de código aberto e por possuir excelentes bibliotecas. Para a área de desenho dos diagramas UML é utilizado a biblioteca *Kinetic* pelo seu alto desempenho. Em um contexto próximo Azevedo et al. (2013) utilizam para tecnologia *Back-And* a linguagem Java e os *framework Spring* e *hibernate*, para *Front-End* é utilizado a linguagem de programação *JavaScript*, *cometD* para mensagens do cliente para o servidor e *ExtJS 3* para a biblioteca de interface do usuário. De outra forma, na ferramenta *WebGME*, Maróti et al. (2014) utilizam uma arquitetura baseada no paradigma *Asynchronous Javascript and XML* (AJAX) onde toda a lógica de visualização e parte da lógica de negócio é executada no navegador, o servidor fornece acesso escalável para o banco de dados dos modelos e coordena a colaboração dos clientes com o modelo.

Izquierdo et al. (2013) utilizam a ferramenta *Collaboro* baseada em Eclipse para o desenvolvimento de DSML, adicionando à aplicação o *BottomUp* para indução baseado no exemplo de meta modelos. No mesmo âmbito, o editor de modelos de negócios apresentado por Lins et al. (2011) foi implementado como *plug-in* do Eclipse, estendendo o editor de *Business Process Model and Notation* (BPMN).

3.2.3 Que estratégias são usadas para tratar a edição colaborativa?

Para responder esta questão dividimos as estratégias aplicadas nos trabalhos pela comunicação utilizada, ou seja, comunicação assíncrona e síncrona. Tratando de comunicação síncrona, Osman (2013) utiliza a arquitetura Cliente-Servidor, com *NodeJS* para o tratamento dos acessos simultâneos à ferramenta *Web*. O estilo da colaboração em *Sawa* é inspirado na arquitetura *Croquet*, desenvolvida com foco na colaboração entre equipe. No *Croquet* foi desenvolvida uma nova arquitetura de colaboração chamada *TeaTime*, com o objetivo que cada usuário possa ver no que o outro está trabalhando. No lado servidor é utilizado um roteador para registrar a data, hora e sequência dos comandos dos clientes e repassar para os demais. Já Nicolaescu, Derntl e Klamma (2013) optaram por utilizar a tecnologia *widget* no *SyncLD*. Para a distribuição das edições é utilizado a comunicação *inter-widget* com base em *Extensible Messaging and Presence Protocol* (XMPP), um protocolo aberto para sistemas de mensagens instantâneas que é baseado em XML, para realizar a implementação do protocolo XMPP é utilizado o *Strophe*, um conjunto de bibliotecas *JavaScript*.

Outra estratégia adotada é demostrada por Dollmann et al. (2011), a ferramenta de modelagem de processos de negócios CoMoMod realiza a distribuição entre os modeladores através de conexões *peer-to-peer*, desta forma cada instância do sis-

tema possui as mesmas funcionalidades e operações dos demais. De forma diferente, com os modeladores trabalhando juntos no mesmo ambiente, a ferramenta coBPM trata a colaboração através de um dispositivo móvel, com uma grande tela sensível ao toque, utilizada pela equipe para construir os modelos em conjunto (DÖWELING et al., 2013).

Referindo-se à comunicação assíncrona, uma das estratégias utilizadas é apresentada por Jiang, Zhang e Zhao (2014), utilizando a abordagem de modelagem colaborativa baseada em estigmergia, onde vários modeladores participam por interações indiretas. Seu princípio se baseia em que uma modificação realizada por um modelador no ambiente colaborativo estimula a ação de outra modificações, pelo mesmo modelador ou por outro. De um modo diferente, Maróti et al. (2014) utilizam a comunicação assíncrona implementando com uma biblioteca *JavaScript*, para realizar a troca de informações com o banco de dados do servidor utilizando *WebSockets*, é utilizado o mesmo código no navegador e no servidor em um recipiente *Node.js*. Por fim, Lins et al. (2011) utilizam a estratégia de colaboração através da nuvem, onde a ferramenta foi implementada em *Eucalyptus*, um ambiente de nuvem *open-source* que prevê, entre vários recursos, escalabilidade e ferramentas de gerenciamento de nuvem.

3.3 Lições do capítulo

Os trabalhos que foram apresentados neste capítulo descrevem ferramentas de modelagem que contemplam as questões de pesquisa do mapeamento. As questões nos ajudaram na definição de algumas decisões da ferramenta, como a plataforma, que a maior parte foram projetadas para a plataforma *Web* (NICOLAESCU; DERNTL; KLAMMA, 2013; JIANG; ZHANG; ZHAO, 2014; OSMAN, 2013; AZEVEDO et al., 2013; MARÓTI et al., 2014; COGNINI et al., 2013) tendo como justificativa a portabilidade e a facilidade de uso, devido a não necessitar realizar *download* e instalação. Também ajudaram na decisão da comunicação a ser adotada na ferramenta, sendo que a maioria utiliza comunicação síncrona (NICOLAESCU; DERNTL; KLAMMA, 2013; DOLLMANN et al., 2011; DÖWELING et al., 2013; OSMAN, 2013), nos influenciando a também utiliza-la.

Outra questão frequente nos trabalhos são os conflitos dos modelos, onde são propostas formas de tratá-los e evitá-los (MICHAUX et al., 2011; NICOLAESCU; DERNTL; KLAMMA, 2013; JIANG; ZHANG; ZHAO, 2014; OSMAN, 2013; MARÓTI et al., 2014; DÖWELING et al., 2013; AZEVEDO et al., 2013). Os trabalhos que não respondem às questões de pesquisa também foram importantes, como por exemplo, a abordagem para resolver conflitos em ferramentas de modelagem colaborativa (BROSCH et al., 2012).

O trabalho que mais se destacou foi o desenvolvido por Osman (2013), no

qual demonstra uma ligação direta com esta presente pesquisa. O autor realiza o desenvolvimento de uma ferramenta de modelagem colaborativa de diagramas UML, porém, ela não segue totalmente a especificação UML, algumas restrições não foram incluídas, pois Osman (2013) acredita que elas atrapalham na produtividade dos desenvolvedores.

4 FERRAMENTA DE MODELAGEM COLABORATIVA

Neste capítulo descrevemos o desenvolvimento da ferramenta de modelagem colaborativa proposta. Na Seção 4.1 justificamos a sua criação. Na Seção 4.2 apresentamos a visão geral de solução para a ferramenta. Na Seção 4.3 demonstramos como foi realizado a análise, projeto e a codificação, apresentando alguns artefatos gerados durante o desenvolvimento da ferramenta. Na Seção 4.5 são abordados as lições desse capítulo.

4.1 Justificativa

A colaboração é um dos pilares do desenvolvimento de *software*, independente da abordagem utilizada. Para os desenvolvedores abstraírem o problema e as funcionalidades desejadas de determinado cliente, é necessário o trabalho colaborativo entre ambos. Com o escopo especificado, os desenvolvedores precisam trabalhar em conjunto para criar uma solução que atenda as necessidades do cliente. Larman (2007) apresenta uma forma, onde vários desenvolvedores desenham diagramas em um ou mais quadros brancos colaborativamente. Bartelt, Vogel e Warnecke (2013) relatam o uso de dispositivos móveis sensíveis ao toque, em conjunto com uma ferramenta para desenhar os diagramas. Em ambas as formas apresentadas pelos autores Larman (2007), Bartelt, Vogel e Warnecke (2013), os desenvolvedores precisam trabalhar no mesmo ambiente físico.

Quando os desenvolvedores não trabalham no mesmo ambiente físico, é necessário usar mecanismos para trabalhar em colaboração com os demais membros da equipe, como o uso de *softwares* de compartilhamento de tela, arquivos e áudios em conjunto com ferramentas de desenho, redes sociais e *e-mails* (OSMAN, 2013; BARTELT; VOGEL; WARNECKE, 2013).

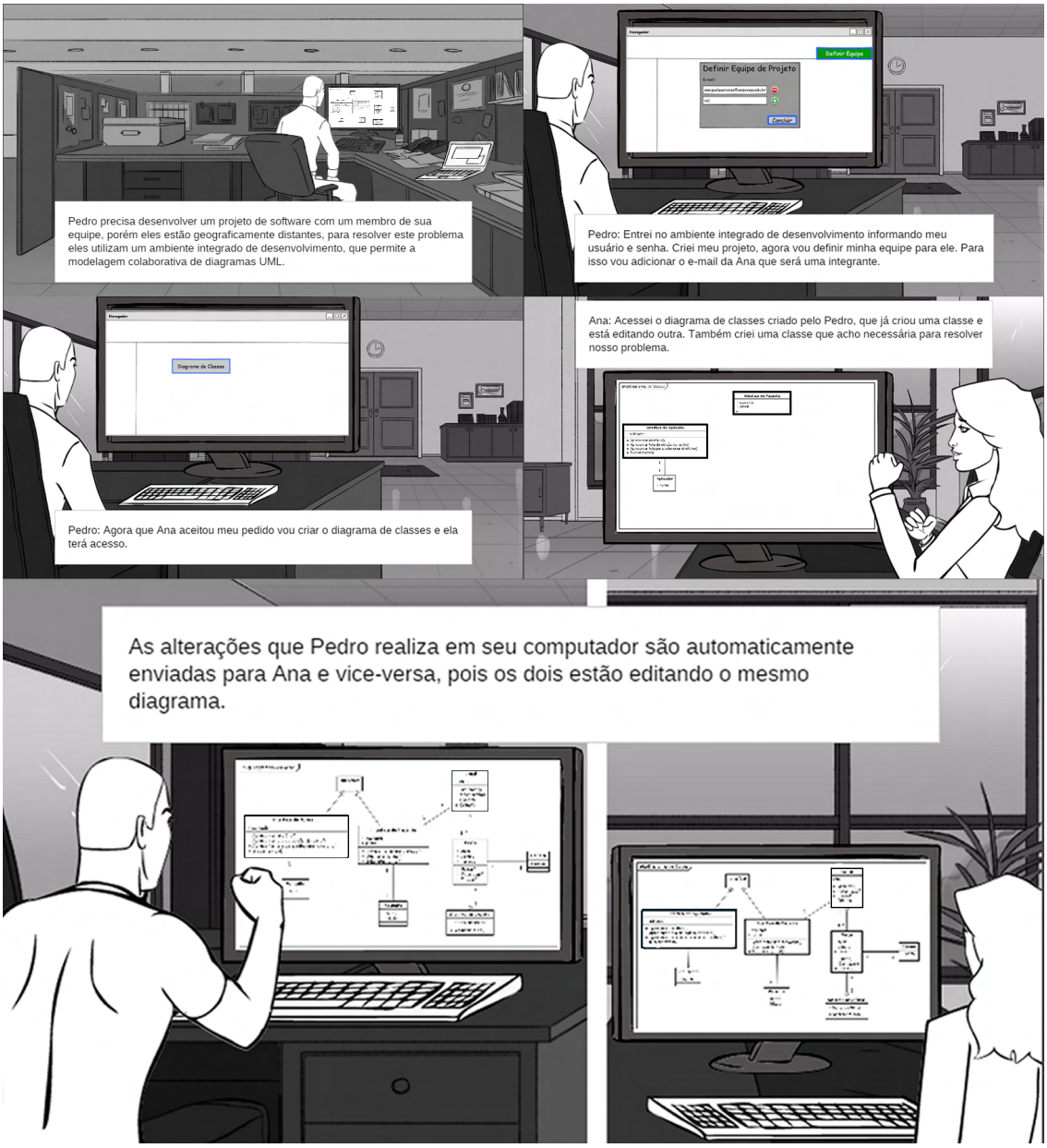
Essas formas de trabalho colaborativo segundo Osman (2013) afetam a produtividade da equipe de desenvolvimento negativamente e podem atingir a qualidade do produto final. Além disso, como existem poucas ferramentas com o objetivo de dar suporte a modelagem colaborativa, os desenvolvedores não ficam com boas alternativas de auxilio na criação dos modelos concorrentemente.

4.2 Visão geral de solução

Para entendermos melhor as funcionalidades de uma ferramenta que provê a modelagem colaborativa e ter uma visão geral do seu funcionamento, utilizamos a

técnica de *storyboard* [1]. Criamos uma história descrevendo os passos que desenvolvedores realizariam caso possuíssem uma ferramenta que promove a modelagem colaborativa, demonstrada na Figura 8.

Figura 8 – *StoryBoards* da Ferramenta.

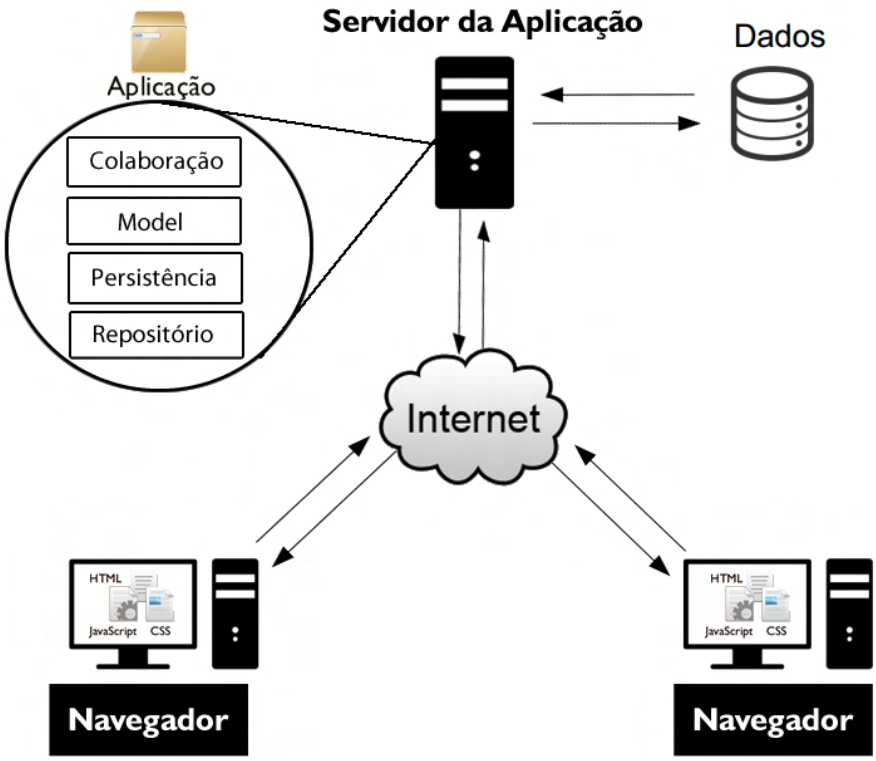


Com o cenário exposto na Figura 8, podemos perceber algumas funcionalidades desejadas em uma ferramenta de modelagem colaborativa, como criar e compartilhar projetos, criar e atualizar diagramas, criar classes, métodos, atributos e relações, entre outras funcionalidades.

[1] Storyboard é uma técnica utilizada para obter um panorama geral do sistema, identificando suas principais funcionalidades e os atores que interagem com ele, de acordo com a narrativa dos personagens.

Na Figura 9 apresentamos uma visão geral de solução para a ferramenta. Cada desenvolvedor acessa a ferramenta através do navegador. Realiza sua autenticação, após isso, passa a ter acesso as funcionalidades da ferramenta. A aplicação exibe suas informações através do *HTML*, *JavaScript* e *CSS* no navegador do usuário.

Figura 9 – Visão geral de solução da Ferramenta.



4.3 Análise e projeto

Com a aplicação da técnica de *storyboards*, definimos algumas funcionalidades levando em consideração a perspectiva do usuário. Para especificar as funcionalidades utilizamos histórias de usuários[2] que demostram na visão dos usuários as funções que o sistema deve exercer. Apresentamos as histórias na Tabela 2. Após definir a primeira versão da visão geral da ferramenta exibida na Figura 9, montamos a sua arquitetura e dividimos em: *view*, *beans*, persistência, repositório, comunicação, *model* e utilitário, como é possível visualizar na Figura 10. A camada *view* tem como responsabilidade a apresentação das páginas web e da representação do diagrama de classes, com as classes, atributos, métodos e as relações, através da biblioteca *JavaS-*

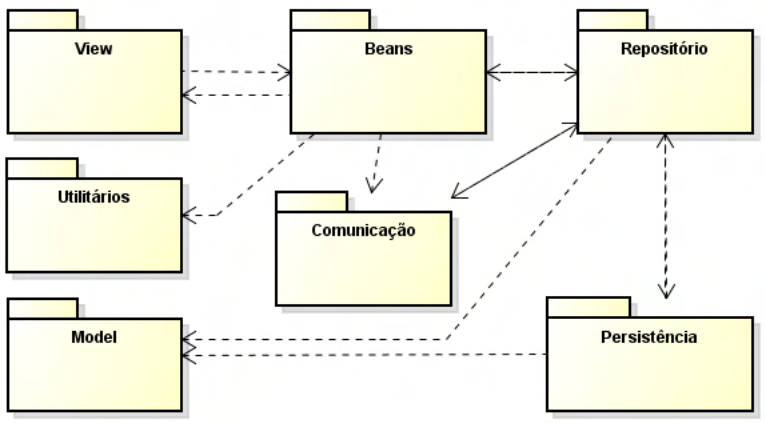
[2] Histórias de usuários: descrevem em uma simples frase a visão do *software* nas perspectivas dos usuários, ou seja, em uma frase simples e objetiva é descrita as funcionalidades que um determinado sistema deve realizar aos olhos do usuário.

Tabela 2 – Histórias de Usuários.

ID	Descrição
01	Como desenvolvedor, eu quero acessar a ferramenta informando meu usuário e senha para que eu tenha acesso às suas funcionalidades.
02	Como desenvolvedor proprietário, eu quero criar projetos para compartilhá-los e adicionar diagramas de classes a ele.
03	Como desenvolvedor proprietário, eu quero compartilhar os meus projetos com outros desenvolvedores para que todos tenham acesso aos diagramas de classes que nele estiverem e que serão adicionado posteriormente.
04	Como desenvolvedor, eu quero criar diagramas de classes para que todos os desenvolvedores tenham acesso a ele.
05	Como desenvolvedor, eu quero criar classes que sejam automaticamente atualizadas para os desenvolvedores que participam do projeto.
06	Como desenvolvedor, eu quero criar métodos e atributos para uma classe e que seja automaticamente atualizadas para os desenvolvedores que participam do projeto.
07	Como desenvolvedor, eu quero criar relações entre classes que sejam automaticamente atualizadas para os desenvolvedores que participam do projeto.
08	Como desenvolvedor, eu quero editar colaborativamente os diagrama de classes que estão no projeto para que todos tenham acesso às minhas modificações realizadas.

cript, *GoJS*. A camada *beans* é responsável por intermediar a comunicação entre as páginas, além de monitorar os eventos realizados pelos usuários e delegar ações para as demais camadas do projeto. A camada persistência tem como responsabilidade armazenar as informações no banco de dados. A camada repositório é responsável por manter os objetos em tempo de execução e realizar as suas atualizações. A camada de comunicação é responsável por informar aos desenvolvedores as modificações realizadas no modelo, a cada alteração ela faz a distribuição da modificação a todos que pertence à determinado projeto ou diagrama de classes. A camada *model*, separa as responsabilidades referentes à UML. A camada utilitário é responsável por manter as informações de acesso de cada usuário.

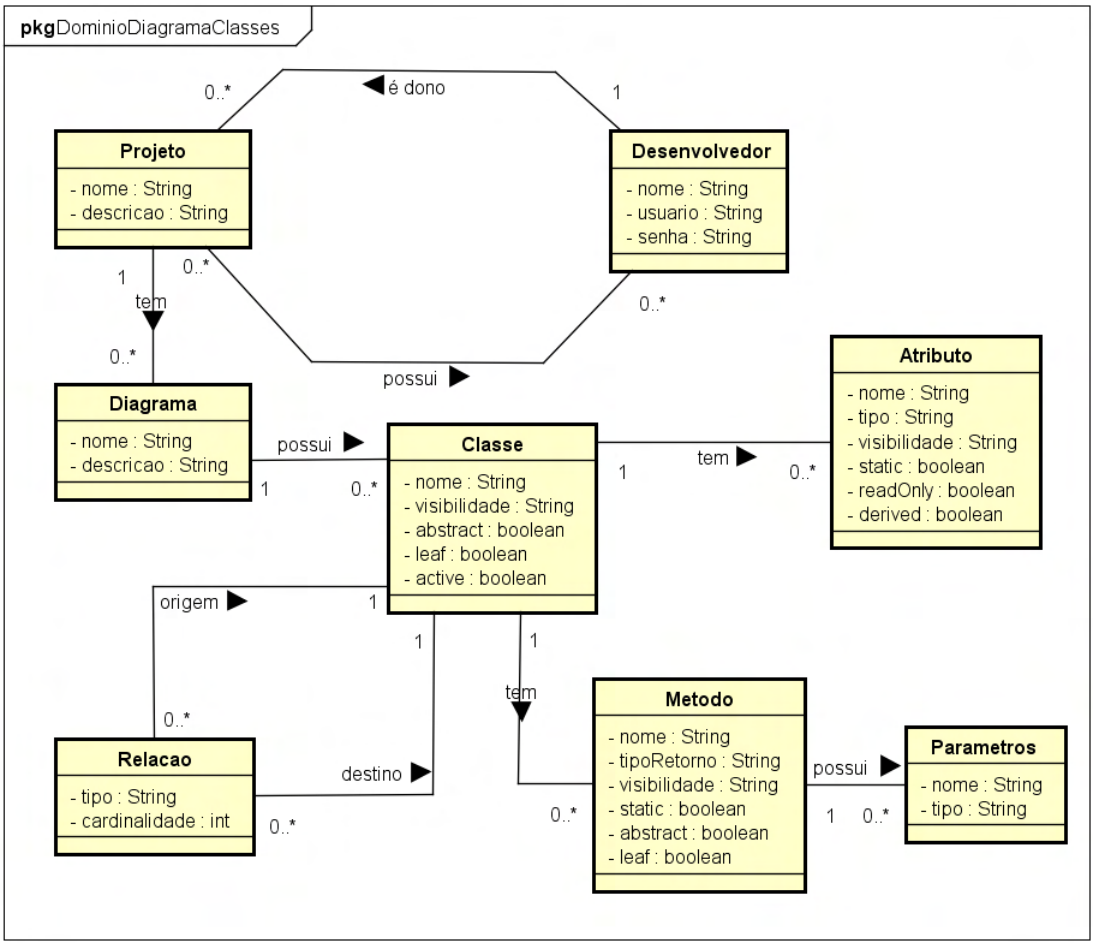
Figura 10 – Arquitetura do *Software*.



4.3.1 Domínio do diagrama de classes

Descrevemos através do digrama da Figura 11, a estrutura do diagrama de classes que a ferramenta suporta. É possível visualizar que um desenvolvedor pode ser dono de um projeto, como também, fazer parte de outros projetos. O projeto pode conter vários ou nenhum diagrama. O diagrama contém zero ou muitas classes, a classe pode ter vários atributos e métodos ou nenhum. A relação tem uma classe de origem e uma classe de destino e a classe pode possuir muitas ou nenhuma relação.

Figura 11 – Diagrama de classes do domínio do diagrama de classes da ferramenta.



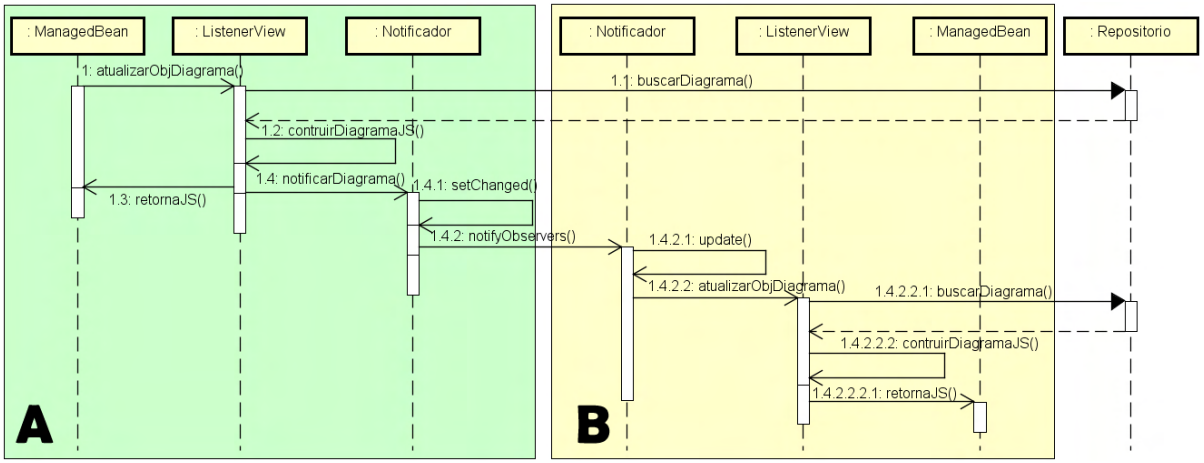
4.3.2 Processo de colaboração

A colaboração é possível através do padrão de projeto *Observer*, que informa as mudanças realizadas no diagrama a todos os desenvolvedores que estão colaborando ao mesmo tempo. Cada usuário possua os objetos "*listenerView*" e "Notificador". O objeto "notificador" é o responsável por enviar e receber as informações, ou seja, ele faz o papel de *Observer* e *Subject* de acordo com o padrão de projeto *Observer*. O

objeto “*listenerView*” é responsável por atualizar a tela dos usuários seja quando ele faz uma alteração, ou quando outros usuário realizam modificações.

No diagrama de sequência da Figura 12, cada retângulo identificado como “A” e “B” estão representando os objetos de dois usuário. A troca de mensagem que ocorre no diagrama é decorrente de uma alteração que o usuário “A” realizou. Este usuário, atualiza a própria tela, buscando as informações no objeto “Repositorio”. Vale salientar que o objeto “Repositorio” utiliza o padrão de projeto *Singleton*, explicado no Capítulo 2. Logo após, envia uma mensagem para o objeto “notificador”, informando que realizou alguma modificação no diagrama. Quando o usuário “B” recebe a notificação é chamado o “update()” que dispara a atualização do seus objetos, sendo assim, são buscadas as modificações no objeto “Repositorio” e posteriormente atualizado os dados do usuário “B”.

Figura 12 – Troca de mensagens dos objetos de dois usuários colaborando em um mesmo diagrama.



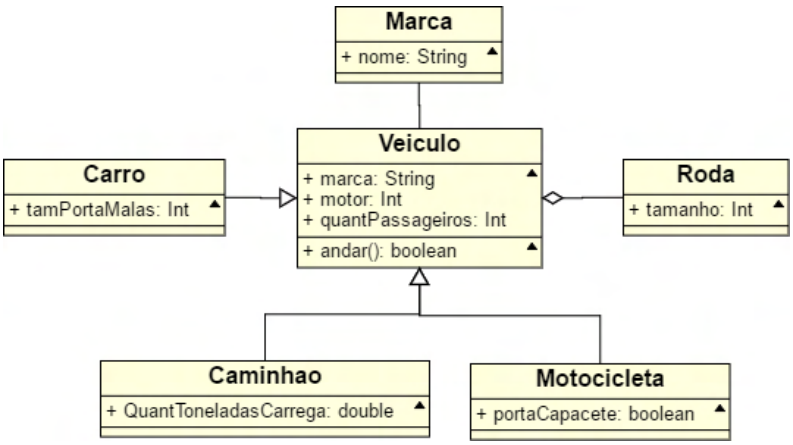
4.4 Detalhes da implementação

Nesta seção vamos apresentar algumas evidências da implementação da ferramenta, como a criação de classes de forma colaborativa, compartilhamento de projeto, criação de associações entre classes, criação de atributos e métodos.

4.4.1 Utilização da biblioteca *GoJS* na ferramenta

Para realizar a modelagem de digramas de classes, é necessário representá-los graficamente. Utilizamos a biblioteca *GoJS* para exibir as classes, métodos, atributos e associações entre classes. Podemos ver na Figura 13 um exemplo de diagrama de classes criado na ferramenta.

Figura 13 – Exemplo de diagrama da ferramenta



As informações que são exibidas chegam no formato *JavaScript Object Notation* (JSON) e são distribuídas nas suas respectivas localidades. Quando ocorre alguma modificação no diagrama, as informações são enviadas para o servidor no mesmo formato, dessa forma a entrada e saída de informações do diagrama tem o mesmo padrão.

No Código 4.1 é possível visualizar o mesmo diagrama da Figura 13 no formato JSON. As informações das classes estão dentro dos colchetes logo após a palavra *"nodeDataArray":* na segunda linha, que representa um vetor com objetos. Os objetos neste caso são as classes. As classes possuem seus atributos e métodos, representados respectivamente por *"properties"* e *"methods"*.

Código 4.1 – Diagrama no formato JSON.

```
{ "class": "go.GraphLinksModel",
"nodeDataArray": [ {"key":"30", "loc":"-103 25", "name":"Veiculo",
   "properties":[{"id":"16", "name":"marca", "type":"String",
   "visibility":"public"}, {"id":"18", "name":"motor", "type":"Int",
   "visibility":"public"}, {"id":"20", "name":"quantPassageiros",
   "type":"Int", "visibility":"public"} ], "methods":[{"id":"17",
   "name":"andar", "type": "boolean", "visibility":"public"} ]},
   {"key":"31", "loc":"-303 46", "name":"Carro",
   "properties":[{"id":"23", "name":"tamPortaMalas", "type":"Int",
   "visibility":"public"} ]}, {"key":"32", "loc":"-237 183",
   "name":"Caminhao", "properties":[{"id":"22", "name":
   "QuantToneladasCarrega", "type":"double", "visibility":"public"}
   ]}, {"key":"33", "loc":"119 46", "name":"Roda",
   "properties":[{"id":"17", "name":"tamanho", "type":"Int",
```

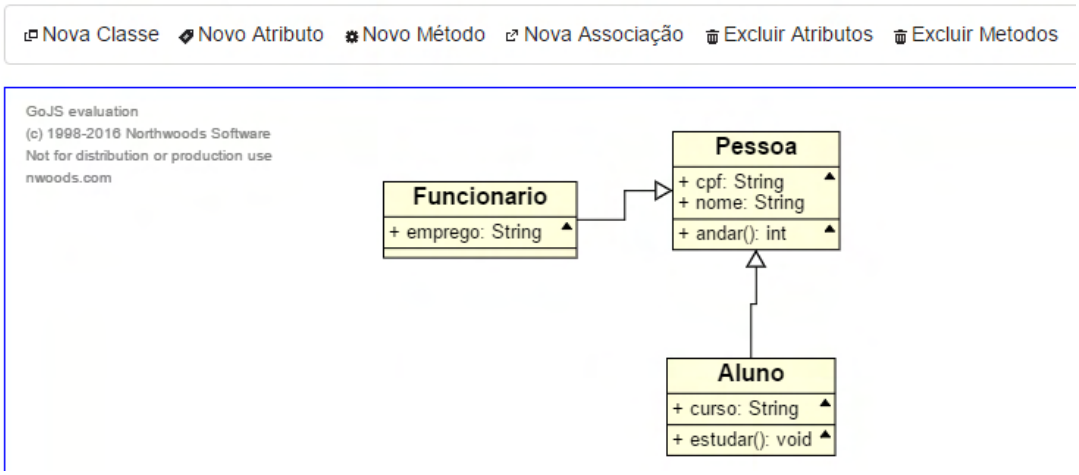
```
    "visibility":"public"} ]}, {"key":"34", "loc":"15 185",
    "name":"Motocicleta", "properties":[{"id":"24", "name":
    "portaCapacete", "type":"boolean", "visibility":"public"} ]},
    {"key":"35", "loc":"-77 -58", "name":"Marca",
    "properties":[{"id":"19", "name":"nome", "type":"String",
    "visibility":"public"} ]} ],
"linkDataArray": [ {"from":"31", "to":"30",
    "relationship":"generalization", "id":"22"}, {"from":"32",
    "to":"30", "relationship":"generalization", "id":"23"},
    {"from":"33", "to":"30", "relationship":"aggregation", "id":"26"},
    {"from":"34", "to":"30", "relationship":"generalization",
    "id":"28"}, {"from":"35", "to":"30", "relationship": "association",
    "id":"29"}]
}
```

Nas ultimas linhas do Código 4.1, estão as relações entre as classes. Dentro dos colchetes, após a palavra *"linkDataArray":*. Estas informações, representam os dados exibidos visualmente no diagrama de classes da Figura 13.

4.4.2 Modelagem de diagrama de classes na ferramenta

Na modelagem do diagrama na ferramenta, podemos realizar várias ações como, criar classes, relações, atributos e métodos. Essas opções estão presentes em um menu superior, como mostra a Figura 14. Os elementos do modelo de classes criados são exibidos logo abaixo do menu. As edições nos elementos que estão no diagrama é realizada diretamente no respectivo modelo.

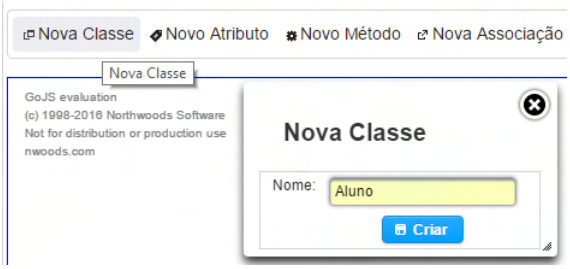
Figura 14 – Menu da área de modelagem.



Considere dois desenvolvedores trabalhando em um mesmo projeto, no mesmo diagrama e ao mesmo tempo. Agora vamos descrever algumas funcionalidades, mostrando como é realizado o funcionamento interno da ferramenta.

Para criar uma classe, o usuário seleciona a opção “Nova Classe”, preenche o nome da classe e pressiona “Criar”, como mostra a Figura 15. O usuário não pode deixar o nome da classe em branco e nem criar classes com o mesmo nome, caso ele tente realizar umas dessas ações é exibida uma mensagem de aviso.

Figura 15 – Exemplo de criação de classe.



Quando o usuário realiza essa operação, um objeto é criado e adicionado na Repositório. A nova classe é exibida na tela do usuário através da biblioteca *GoJS* e posteriormente enviada para os demais usuários, como mostrado no diagrama de sequência da Figura 12.

Na Figura 16 mostramos de forma mais simples a colaboração entre dois usuário, evidenciando a comunicação entre os objetos dos dois usuários para estabelecer a colaboração. A Figura 16 exibe a colaboração a partir do momento que a classe que o usuário “A” criou já está em sua tela.

É possível criar atributos e métodos para as classes, a Figura 17 exibe as telas de criação. Para criar tanto métodos, quanto atributos, sendo que não é possível deixar qualquer campo em branco.

Além de atributos e métodos, é possível criar relações entre as classes, dentre elas estão, associação, agregação, composição e generalização. Exibimos na Figura 18 a tela pra criar relações.

Tanto na criação de atributos como métodos e relações, a comunicação ocorre da mesma forma exibida no diagrama de sequencia da Figura 12 e de acordo com o exemplo de alto nível exibido na Figura 16. Independente do elemento que for criado no modelo a forma de comunicação ocorre da mesma maneira para todos.

As edições dos elementos do diagrama ocorre de uma forma pouco diferente, ao invés de atualizar diretamente o objeto “repositório”, as informações são alteradas primeiro no diagrama do usuário que o modificou. Após isso, as alterações são envi-

Figura 16 – Ilustração da colaboração entre dois usuários.

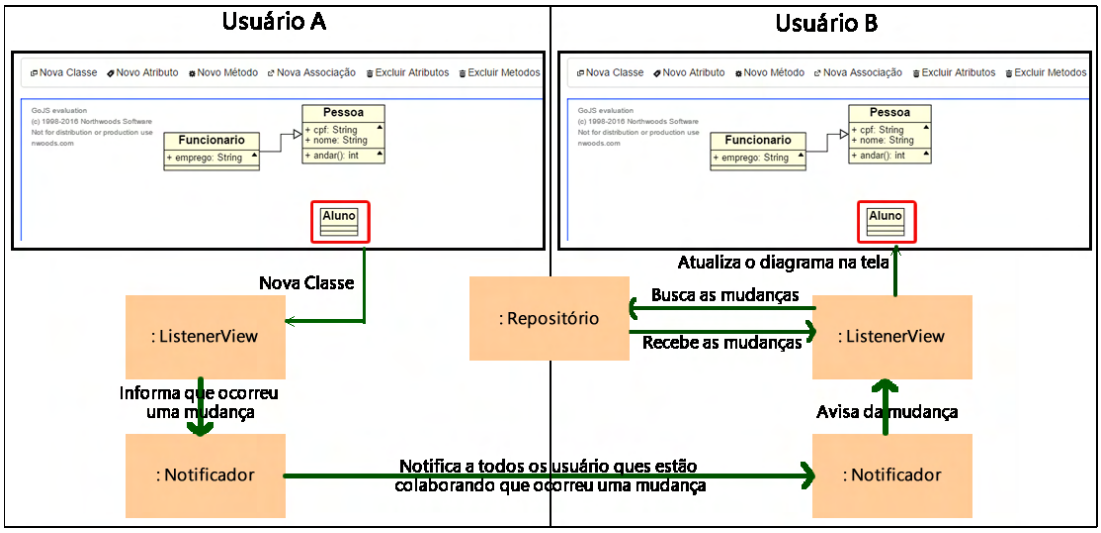


Figura 17 – Telas para criar atributos e métodos respectivamente.

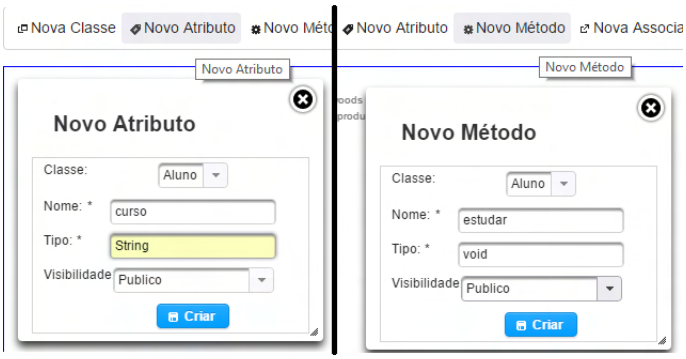
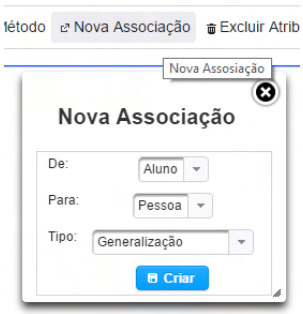


Figura 18 – Tela para criar relações.



adas no formato JSON para o “repositório”, que transforma em objeto e verifica quais foram as mudanças. No decorrer a comunicação segue igual à exibida na Figura 16.

4.5 Lições do capítulo

Demonstramos nesse capítulo a construção da ferramenta de acordo com a justificativa que apresentamos. Detalhamos todo o processo para criar uma solução que atenda as necessidades abordadas. Conseguimos desenvolver um produto simples e funcional, para dar suporte a modelagem colaborativa de diagramas de classe. Para definir as tecnologias que utilizamos na ferramenta, testamos elas, verificando se eram adequadas para o nosso propósito.

A colaboração que é o principal objetivo da ferramenta, conseguimos resolver de uma forma simples utilizando o padrão de projeto *Obverser*. Desta maneira, chegamos a uma versão estável da ferramenta.

5 TESTES E VALIDAÇÃO

Neste capítulo apresentamos os testes realizados na ferramenta e sua validação. Na Seção 5.1 detalhamos quais testes foram utilizamos e os seus resultados. Na Seção 5.2 mostramos o teste realizado com possíveis usuários da ferramenta e como foi realizada a avaliação. Na Seção 5.3 apresentamos as lições do capítulo.

5.1 Testes

Para os testes da ferramenta, optamos por fazer testes funcionais manuais. Esta escolha foi baseada de acordo com o tempo restante para a finalização deste trabalho e por avaliar se a ferramenta está em conformidade com as funcionalidades que definimos nas etapas anteriores do desenvolvimento.

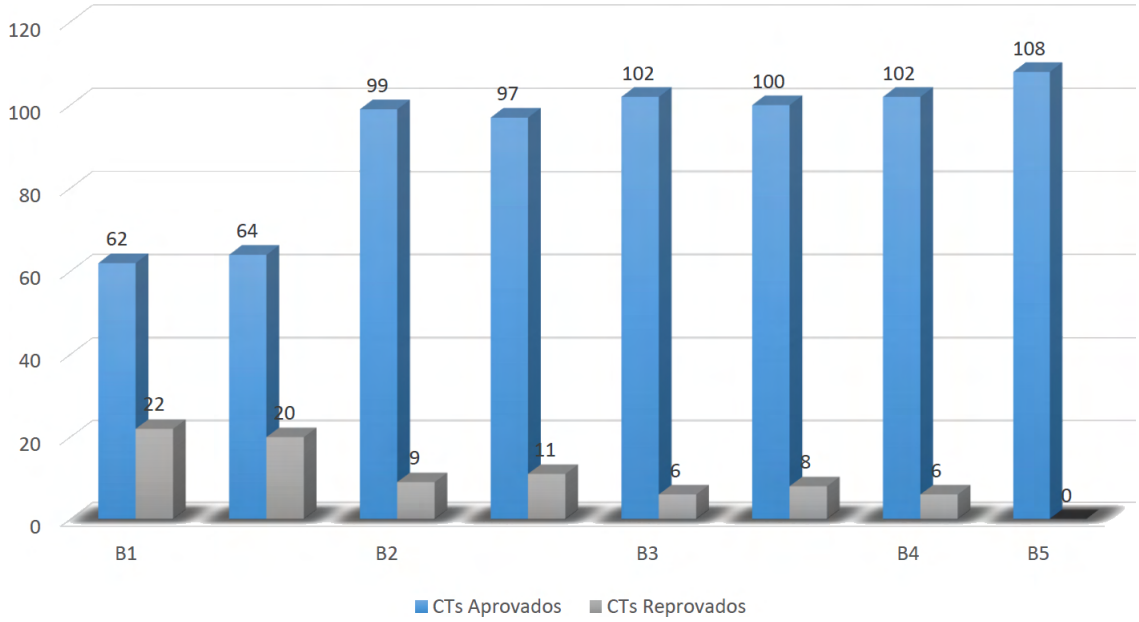
Os testes foram planejados para encontrar falhas nas funcionalidades da ferramenta e corrigir. Sendo assim, realizamos cinco baterias de testes, com uma média de 95 casos de testes em cada bateria. As três primeiras baterias foram realizadas por dois testadores e as duas últimas por apenas um em cada. Iniciamos na primeira bateria de testes com 84 casos de testes e finalizamos com 108 casos. Na Figura 19 é exibido um exemplo de caso de teste utilizado. Cada caso de teste possui um identificador, suas pré, e pós-condições, os dados de entrada para realizar o teste, o fluxo que deve ser seguido e os resultados esperados. Se o caso de teste retornar os resultados esperados, ele passou, caso contrário, ele reprovou no teste.

Figura 19 – Exemplo de um casos de teste.

Identificador:	CT 104
Pré- condições:	Sistema possuir uma classe já cadastrada com um ou mais usuário online editando o diagrama
Pós-condições:	
Dados de Entrada	
Classe: Pessoa, nome: espaços em branco, tipo: espaços em branco e Visibilidade: Protected	
Fluxo	
1	Clicar em "Novo Método".
2	Selecionar Classe Pessoa
3	Seleciona a visibilidade como "Protected"
4	Define o nome como espaços em branco e o tipos qualquer.
5	Clicar em Salvar
6	Exibe mensagem de erro
Resultados Esperados	
1	Exibe a tela de Criar Método
2	Operação do usuário
3	Operação do usuário
4	Operação do usuário
5	Verifica os dados de entrada e encontra dados em brancos.
6	Exibe mensagem de Nome e Tipo não podem estar em brancos

Sintetizamos os resultados das baterias de teste no gráfico da Figura 20. As três primeiras baterias(B1,B2 e B3) foram realizadas por dois testadores, ou seja, cada testador realizou ela uma vez e as duas últimas por um testador cada. É possível notar pouca variança de defeitos encontrados por cada testador, vale ressaltar, que um dos testadores é o autor deste trabalho e o outro um aluno voluntário do 6º semestre do curso de Engenharia de *software* da Universidade Federal do Pampa.

Figura 20 – Gráfico do resultado das baterias de testes.



Podemos ver na Figura 20 que nas primeiras baterias foram encontrados a maior quantidade de defeitos e gradualmente foi diminuindo nas baterias posteriores. Isso ocorre porque ao final de cada bateria ocorreu a correção dos erros e uma nova bateria de testes era executada, até solucionar todos os erros encontrados. Dessa forma, foram necessárias 5 baterias de teste até ter todos os erros corrigidos. Para título de informação, as baterias de testes foram executadas em ambiente de desenvolvimento.

5.2 Validação

Depois de realizar a correção dos erros na etapa de desenvolvimento e chegar a uma versão estável da ferramenta, ocorreu uma nova bateria com possíveis usuários para validar a nossa solução. Para isso, implantamos a ferramenta em um servidor local e os usuários acessavam através de seus computadores a ferramenta.

Os usuários que participaram do teste de validação foram 6, todos do curso de Engenharia de *Software* da Universidade Federal do Pampa e que estão cursando

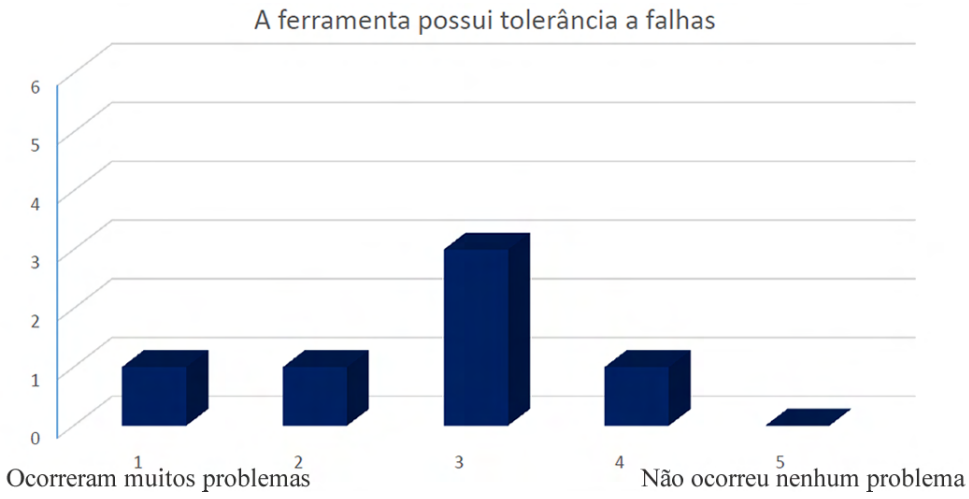
entre o 6º ao 8º semestre do curso. Salientamos que todos os usuário participaram de forma voluntária.

Os usuário realizaram três atividades no teste, as atividades envolviam a criação/edição/exclusão de projetos, diagramas, classes, atributos, métodos, relações e o compartilhamento de projetos com os demais usuários que participaram. Na primeira atividade o usuário realizava essas operações com somente ele editando o diagrama, nas duas seguintes aumentava o numero de usuário editando o mesmo diagrama.

Após realizarem as atividades propostas, os usuários responderam a um questionário de avaliação da ferramenta, baseado na ISO 9126-1 (2003) que prevê as normas brasileiras de qualidade de *software*, especificamente seguindo o modelo de qualidade interna e externa. Este modelo é dividido em seis características que possuem suas sub-características. Entre as seis, optamos por 5 delas no questionário: funcionalidade, confiabilidade, usabilidade, eficiência e portabilidade. Escolhemos as sub-características que se adequaram ao contexto da ferramenta. Para cada sub-característica especificamos uma métrica, que é o que foi avaliado pelos usuários. Como utilizamos a escala *Likert*[1] para obter a opinião dos usuário, as métricas são frases afirmativas. Os usuários em suas respostas informam seu nível de concordância, definimos o nível de um à cinco para o nível de concordância, sendo um discordo totalmente e cinco concordo totalmente. Em outras palavras, quanto maior o valor informado pelo usuário, mais positiva é sua resposta e maior o nível de qualidade.

A Figura 21 apresenta a avaliação dos usuários de acordo com a métrica estabelecida, onde a maior parte dos resultados foram avaliados como intermediários.

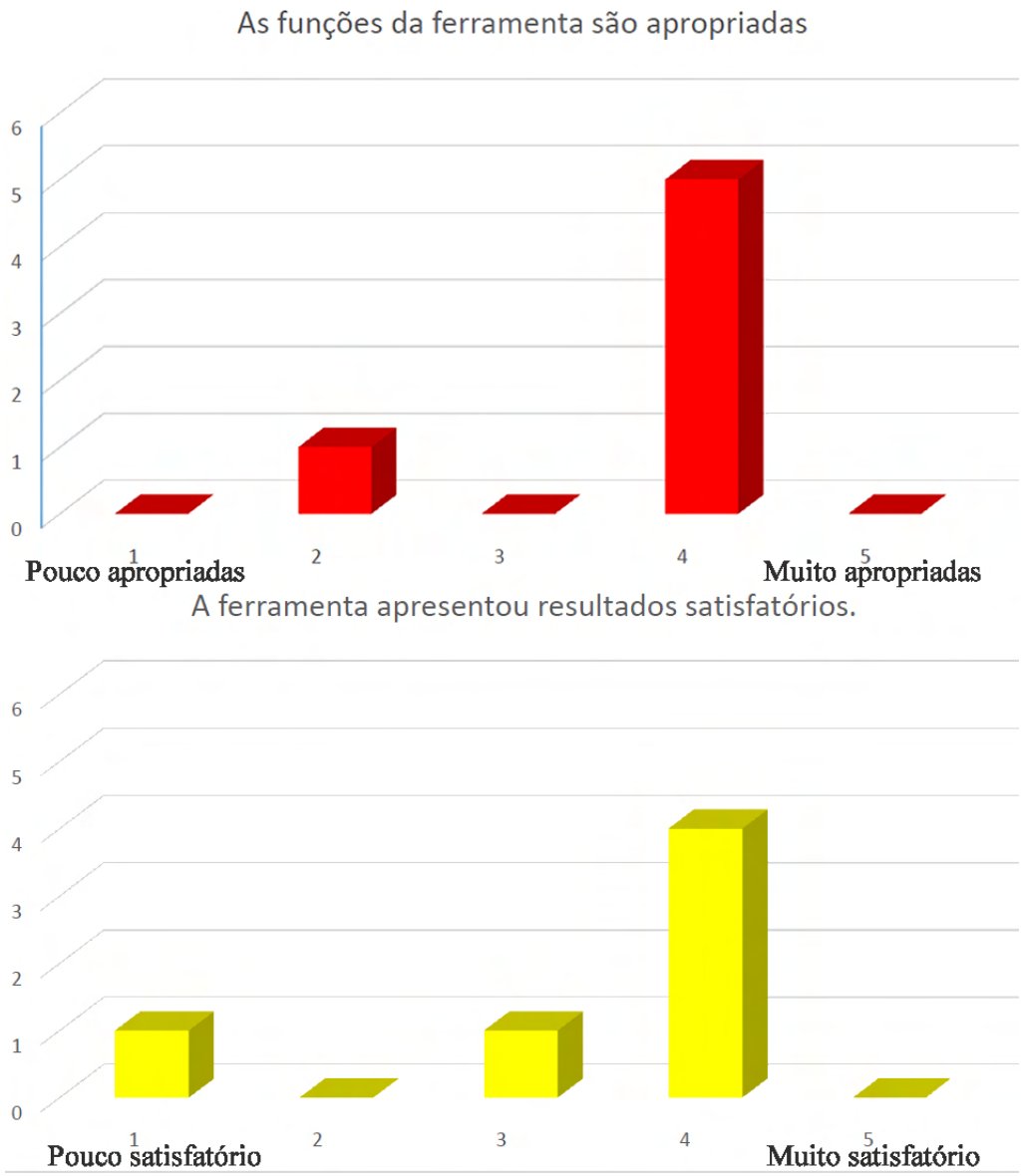
Figura 21 – Gráfico da métrica de confiabilidade.



[1] Escala *Likert* é uma escala de resposta usada em questionários, bastante usada em pesquisas de opinião. Ao responder, os perguntados definem seu nível de concordância com a afirmação imposta (LIKERT, 1932).

Na Figura 22 exibimos os resultados das métricas de funcionalidade, podemos perceber que segundo os usuários que participaram da validação obtivemos resultados positivos, sendo o nível de concordância mais escolhido foi o quatro, por cinco usuários.

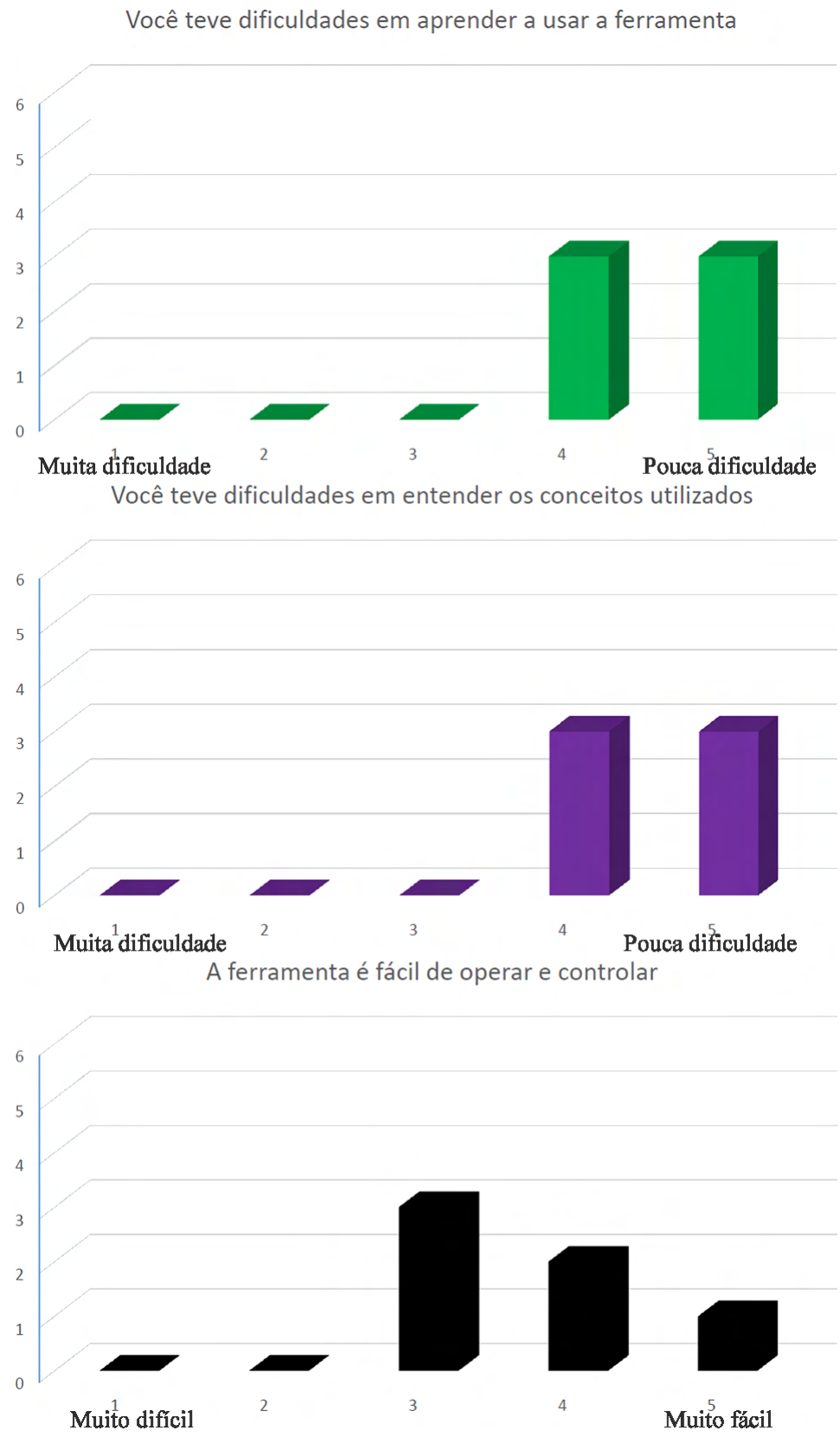
Figura 22 – Gráficos das métricas de funcionalidade.



A Figura 23 demonstra os resultados das métricas de usabilidade, é possível perceber que os resultados foram muito bons, sendo o nível de concordância de cinco dos seis usuários foi de três para cima. Podemos afirmar que a ferramenta dispõe de uma boa usabilidade de acordo com a avaliação realizada.

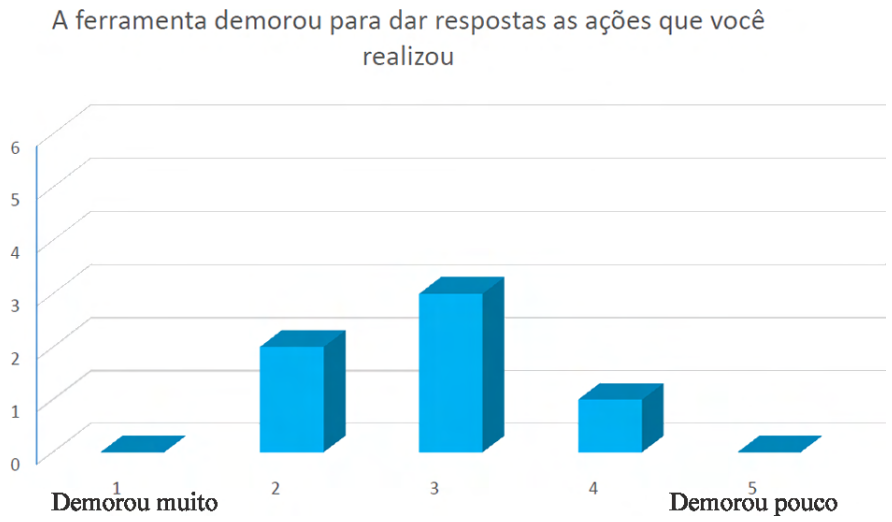
A eficiência da ferramenta de acordo com os usuários é considerada mediana, levando em consideração os resultados apresentados na Figura 24, onde o nível de

Figura 23 – Gráficos das métricas de usabilidade.



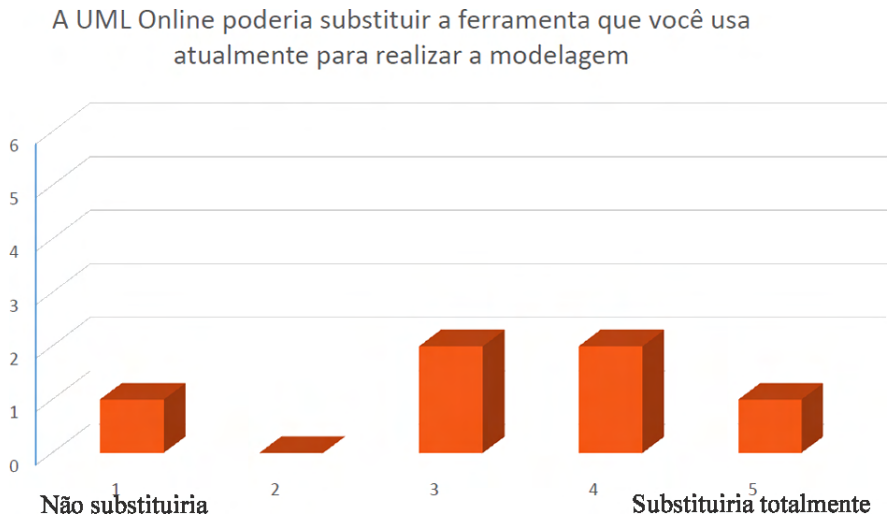
concordância de cinco participantes foi entre dois a três.

Figura 24 – Gráfico da métrica de eficiência.



Em relação a portabilidade da ferramenta, podemos afirmar que ela obteve resultados positivos, de acordo com a Figura 25, o nível de concordância da maior parte dos usuários em relação a afirmação, foi entre três e quatro, sou seja, está de mediana à boa.

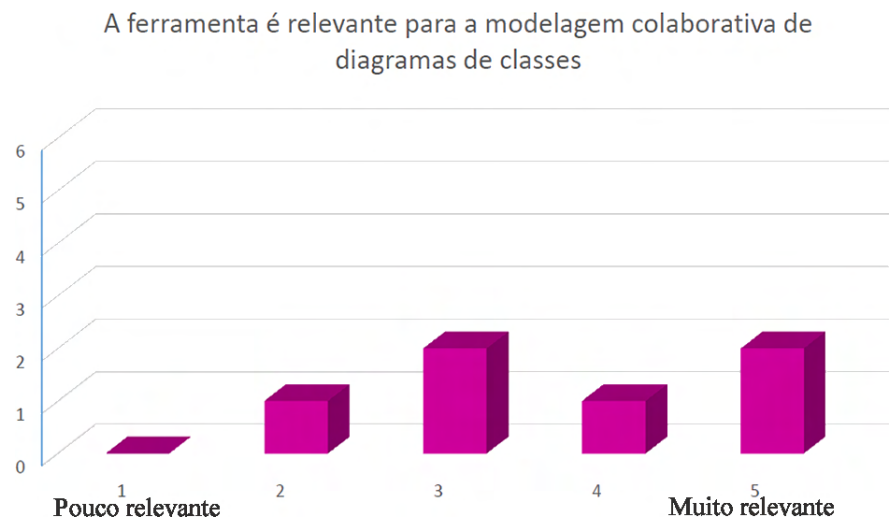
Figura 25 – Gráfico da métrica de portabilidade.



Além da utilização das métricas definidas seguindo a 9126-1 (2003), definimos outra métrica para verificar a relevância da ferramenta para o desenvolvimento de diagramas de classes colaborativamente. Exibimos na Figura 26 o resultado segundo a opinião dos usuários. Podemos perceber que a avaliação foi positiva, sendo que o nível de concordância, com relação a frase afirmativa, de grande parte dos usuário

foi entre três e cinco, sendo assim nossa ferramenta é considerada relevante pelos participantes.

Figura 26 – Gráfico da relevância da ferramenta.



Além dos dados coletados de acordo com a ISO 9126-1 (2003), definimos um campo opcional para recolher a opinião dos usuários. De acordo com as respostas obtidas as coisas que mais se destacaram foi o surgimento de algumas falhas durante o funcionamento da ferramenta, mas que segundo os usuários não atrapalhou o trabalho colaborativo, além disso, outra questão evidenciada foi a lentidão em algumas operações. Por fim, algumas sugestões de melhorias foram abordadas, como um bate papo na área de modelagem, histórico de edições e a possibilidade de visualizar o que cada usuário está editando.

5.3 Lições do capítulo

As etapas de verificação e validação são partes essenciais no desenvolvimento de *software*, durante a verificação encontramos muitos problemas na ferramenta que ocasionavam o seu mal funcionamento, de acordo com os casos de testes que definimos conseguimos corrigir todos os erros que encontramos.

Com a correção dos erros chegamos a primeira versão estável da ferramenta e realizamos a validação. Nesta etapa, conseguimos obter a opinião dos usuários em relação a utilização da ferramenta. Apesar de ocorrer alguns erros durante a validação, o servidor local não ser o mais apropriado e apenas seis usuários participarem da validação por questões de disponibilidade, os usuários demonstraram uma boa aceitação e o teste foi muito gratificante.

No futuro podem ser realizados testes de validação com a ferramenta hospedada em um servidor online para obtenção de melhores resultados e possivelmente

um experimento com usuários.

6 CONCLUSÕES

O objetivo desse trabalho foi o desenvolvimento de uma ferramenta de modelagem colaborativa de diagramas de classes, para melhorar a colaboração entre membros de equipes de desenvolvimento de *software* e construir um modelo final mais consistente.

Para construirmos a ferramenta realizamos vários estudos, entre eles, a linguagem UML, sistemas distribuídos, arquiteturas de *software*, entre outros. Através desses estudos, definimos como a ferramenta deveria ser e quais tecnologias que utilizaria. com base nisso, desenvolvemos a ferramenta em plataforma *web* para ser acessada por um navegador, utilizando as tecnologias JSF e a biblioteca *JavaSript GoJS* para visualização e o *framework Hibernate* para a persistência dos dados.

Durante o desenvolvimento da ferramenta nos debatemos com diversas dificuldades e desafios. O primeiro foi construir uma forma de colaboração eficiente. Utilizamos o padrão de projeto *Observer* para solucionar esta dificuldade. Ele faz a distribuição das modificações que acontecem no diagrama para os usuários que estão editando o mesmo diagrama, e ao mesmo tempo.

Outro desafio que enfrentamos foi com a representação do diagrama graficamente, primeiro testamos o elemento *diagram* do *PrimeFaces*[1], mas ele não dava suporte para o diagrama de classes. Após isso, realizamos alguns testes com a biblioteca *GoJS* para verificar se fornecia o suporte. Desta maneira, aos estudar sobre ela definimos que era apropriada ao nosso contexto, e flexível para um aumento no escopo na ferramenta futuramente.

Apesar dos desafios e dificuldades que encontramos durante o desenvolvimento do trabalho, a ferramenta apresentou uma resposta boa nos testes e na validação. Mesmo ainda não abrangendo o diagrama de classes completo os usuários demonstraram entusiasmo durante o teste de validação.

6.1 Contribuições

No nosso trabalho desenvolvemos uma ferramenta para dar suporte a modelagem colaborativa de diagramas de classes. Ela abrange a criação, edição e exclusão de projetos, diagramas de classes, classes, atributos, métodos e relações de forma colaborativa.

A ferramenta criada auxilia os desenvolvedores que trabalham à distância na

[1] *Primefaces* é uma biblioteca *open-source* de componentes para JSF.

construção de diagramas de classes em conjuntos com os demais membros, os desenvolvedores podem acompanhar as modificações e a evolução do modelo. A ferramenta suporta os seguintes elementos no diagrama de classes, a classe, atributo, método e as relações de generalização, agregação, composição e associação, sem o uso de multiplicidade.

Além dessas contribuições, a ferramenta contribui para os que querem desenvolver ferramentas colaborativas, levando em consideração que nossa estratégia de colaboração é simples e eficiente. Ela também pode ser continuada, adicionando outros diagramas para dar um suporte ainda maior para a modelagem colaborativa.

6.2 Trabalhos futuros

Para trabalhos futuros pretendemos adicionar os elementos restantes do diagrama de classes da UML, como multiplicidade nas relações, esteriótipos e melhorar a qualidade da colaboração, buscando algorítimos mais eficientes para deixa-lá mais rápida. Pretendemos também tratar a questão da concorrência, quando dois ou mais usuários modificam ao mesmo tempo um elemento do modelo. Algumas sugestões dos usuários que participaram da validação também são interessantes, como manter o histórico das alterações e exibir o que cada usuários está editando no diagrama.

Além dessas melhorias na ferramenta, para trabalhos futuros pretendemos realizar um experimento controlado, com a ferramenta em um ambiente de produção para obter um avaliação melhor das suas funcionalidades e futuramente acrescentar outros diagramas da UML.

REFERÊNCIAS BIBLIOGRÁFICAS

9126-1, I. T. *Engenharia de software - Qualidade de produto - Parte 1: Modelo de qualidade.* 2003. Citado 4 vezes nas páginas 25, 61, 64 e 65.

AZEVEDO, D. et al. On the development and usability of a diagram-based collaborative brainstorming component. *J. UCS*, v. 19, n. 7, p. 873–893, 2013. Citado 3 vezes nas páginas 42, 43 e 44.

BARTELT, C.; VOGEL, M.; WARNECKE, T. Collaborative creativity: From hand drawn sketches to formal domain specific models and back again. In: *MoRoCo@ ECSCW*. [S.l.: s.n.], 2013. p. 25–32. Citado 4 vezes nas páginas 23, 31, 32 e 47.

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. [S.l.]: Elsevier, 2007. Citado 4 vezes nas páginas 23, 27, 28 e 31.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML-GUIA DO USUARIO: TRADUÇAO DA SEGUNDA EDIÇAO*. [S.l.]: Elsevier Brasil, 2000. Citado na página 23.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. CAMPUS - RJ, 2006. ISBN 9788535217841. Disponível em: <http://books.google.com.br/books?id=ddWqxcDKGF8C>. Citado 2 vezes nas páginas 23 e 27.

BROSCH, P. et al. Conflict visualization for evolving uml models. *Journal of Object Technology*, v. 11, n. 3, p. 2–1, 2012. Citado na página 44.

CARDOSO, C. *UML na prática: do problema ao sistema*. [S.l.]: Editora Ciência Moderna LTDA, 2003. Citado na página 27.

COGNINI, R. et al. A collaborative approach to public administrations inter-organizational business processes modelling. In: *EGOV/ePart Ongoing Research*. [S.l.: s.n.], 2013. p. 69–75. Citado 2 vezes nas páginas 42 e 44.

DIRIX, M. Awareness in computer-supported collaborative modelling. application to genmymodel. 2013. Citado na página 31.

DOLLMANN, T. et al. Collaborative business process modeling with comomod-a toolkit for model integration in distributed cooperation environments. In: IEEE. *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on*. [S.l.], 2011. p. 217–222. Citado 3 vezes nas páginas 42, 43 e 44.

DÖWELING, S. et al. Collaborative business process modeling on interactive tabletops. In: CITESEER. *ECIS*. [S.l.], 2013. p. 29. Citado 2 vezes nas páginas 42 e 44.

GAMMA, E. *Padrões de Projetos: Soluções Reutilizáveis*. Bookman, 2009. ISBN 9788577800469. Disponível em: <https://books.google.com.br/books?id=U91CYCqTCgkC>. Citado 2 vezes nas páginas 32 e 33.

HAT, R. *Hibernate Object/Relational Mapping (ORM)*. 2016. <http://hibernate.org/orm/>. Acesso em: 22/05/2016. Citado 2 vezes nas páginas 36 e 37.

IZQUIERDO, J. L. C. et al. Engaging end-users in the collaborative development of domain-specific modelling languages. In: *Cooperative Design, Visualization, and Engineering*. [S.l.]: Springer, 2013. p. 101–110. Citado 2 vezes nas páginas 42 e 43.

JIANG, Y.; ZHANG, W.; ZHAO, H. Stigmergy-based collaborative conceptual modeling. In: IEEE. *Global Software Engineeering Workshops (ICGSEW), 2014 IEEE International Conference on*. [S.l.], 2014. p. 45–50. Citado 2 vezes nas páginas 42 e 44.

LARMAN, C. *Utilizando UML e Padrões*. Bookman, 2007. ISBN 9788560031528. Disponível em: <http://books.google.com.br/books?id=ZHtcynS03DIC>. Citado 6 vezes nas páginas 23, 24, 27, 31, 32 e 47.

LIKERT, R. A technique for the measurement of attitudes. *Archives of psychology*, 1932. Citado na página 61.

LINS, F. et al. Ssc4cloud tooling: An integrated environment for the development of business processes with security requirements in the cloud. In: IEEE. *Services (SERVICES), 2011 IEEE World Congress on*. [S.l.], 2011. p. 53–60. Citado 3 vezes nas páginas 42, 43 e 44.

MARÓTI, M. et al. Online collaborative environment for designing complex computational systems. *Procedia Computer Science*, Elsevier, v. 29, p. 2432–2441, 2014. Citado 3 vezes nas páginas 42, 43 e 44.

MEDEIROS, H. *Padrão de Projeto Observer em Java*. 2016. <http://www.devmedia.com.br/padrao-de-projeto-observer-em-java/26163>. Acesso em: 05/05/2016. Citado na página 33.

MICHAUX, J. et al. A semantically rich approach for collaborative model edition. In: ACM. *Proceedings of the 2011 ACM Symposium on Applied Computing*. [S.l.], 2011. p. 1470–1475. Citado 2 vezes nas páginas 42 e 44.

NICOLAESCU, P.; DERNTL, M.; KLAMMA, R. Browser-based collaborative modeling in near real-time. In: IEEE. *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. [S.l.], 2013. p. 335–344. Citado 3 vezes nas páginas 42, 43 e 44.

NORTHWOODS, S. C. *GoJS*. 2016. <https://gojs.net/latest/index.html>. Acesso em: 01/05/2016. Citado 2 vezes nas páginas 35 e 36.

OMG, O. M. G. *Unified Modeling Language (UML)*. 2015. <http://www.omg.org/spec/UML/2.5/Beta2/>. Acesso em: 22/05/2015. Citado 4 vezes nas páginas 27, 28, 29 e 30.

ORACLE, T. N. *JavaServer Faces Technology*. 2015. <http://www.docs.oracle.com/javaee/7/tutorial/>. Acesso em: 12/11/2015. Citado 2 vezes nas páginas 34 e 35.

OSMAN, H. *Web-Based Collaborative Software Modeling*. Tese (Doutorado) — Università della Svizzera Italiana, 2013. Citado 8 vezes nas páginas 23, 24, 32, 42, 43, 44, 45 e 47.

SCHACH, S. R. *Engenharia de Software-: Os Paradigmas Clássico e Orientado a Objetos*. [S.l.]: McGraw Hill Brasil, 2009. v. 7. Citado 2 vezes nas páginas 30 e 31.

SOMMERVILLE, I. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2007. v. 8. Citado 2 vezes nas páginas 23 e 30.