UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ALEXANDRE DE CARLI
PATRICK CECATO ALBANI

SISTEMA PARA GERENCIAMENTO DA DISCIPLINA DE OFICINA DE PROJETO E DESENVOLVIMENTO DE SOFTWARE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2013

ALEXANDRE DE CARLI
PATRICK CECATO ALBANI

SISTEMA PARA GERENCIAMENTO DA DISCIPLINA DE OFICINA DE PROJETO E DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2013

ATA Nº: **236**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **ALEXANDRE DE CARLI**.

Às 16:00 hrs do dia 26 de fevereiro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Eliane Maria de Bortoli Fávero (Convidada) e Lucilia Yoshie Araki (Convidada), para avaliar o Trabalho de Diplomação do aluno Alexandre de Carli, matrícula 1270478, sob o título **Sistema Web para Gerenciamento da Disciplina de Laboratório de Software**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 16:25 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Eliane Maria de Bortoli Fávero, M.Sc.
Convidada

Profa. Lucilia Yoshie Araki, M.Sc.
Convidada

Profa. Eliane Maria de Bortoli Fávero, M.Sc
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

ATA Nº: **237**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **PATRICK CECATO ALBANI**.

Às 16:00 hrs do dia 26 de fevereiro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Eliane Maria de Bortoli Fávero (Convidada) e Lucilia Yoshie Araki (Convidada), para avaliar o Trabalho de Diplomação do aluno Patrick Cecato Albani, matrícula 980471, sob o título **Sistema Web para Gerenciamento da Disciplina de Laboratório de Software**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 16:25 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Eliane Maria de Bortoli Fávero, M.Sc.
Convidada

Profa. Lucilia Yoshie Araki, M.Sc.
Convidada

Profa. Eliane Maria de Bortoli Fávero, M.Sc
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

ALBANI, Patrick Cecato; DE CARLI, Alexandre. Sistema para gerenciamento da disciplina de Oficina de Projeto e Desenvolvimento de Software. 2013. 69f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2013.

A disciplina de Oficina de Projeto e Desenvolvimento do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco visa simular uma fábrica de software. Permitindo assim, aos alunos, realizar o desenvolvimento de um projeto de software no contexto de uma fábrica de software baseada em processos que são definidos de acordo com modelos de qualidade. Os processos permitem organizar as atividades e verificar o que está sendo realizado, por quem está sendo realizado, quais os recursos utilizados e os resultados pretendidos em qualquer momento do ciclo de vida. Processos também auxiliam a predizer os resultados a serem obtidos e, assim, controles podem ser definidos para a verificação das métricas de qualidade estabelecidas. Considerando que processos auxiliam no atendimento de padrões de qualidade, definir uma fábrica de software a partir de processos passa a ser relevante no desenvolvimento de produtos que atendam os requisitos definidos para o mesmo. Para facilitar o gerenciamento das atividades por parte do professor e a realização das mesmas pelos alunos que estão organizados em equipes, por meio da realização deste trabalho um sistema para o gerenciamento da disciplina de Oficina de Projeto e Desenvolvimento foi desenvolvido. Embora a solução tenha sido criada para uma disciplina de um curso específico, a mesma se aplica para qualquer atividade acadêmica que vise o desenvolvimento de projetos de software e para gerenciamento de fábricas de software. Esse sistema foi implementado utilizando tecnologias para desenvolvimento para *web* destacando-se Java com JavaServer Faces e PrimeFaces para a implementação da interface.

Palavras-chave: Java para *web*. JavaServer Faces. PrimeFaces. Fábrica de software. Processos de software.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

CRUD	*Create, Retrieve, Update and Delete*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
JSF	*JavaServer Faces*
JSP	*JavaServer Pages*
MDD	*Model Driven Development*
MVC	*Model-View-Controller*
ORM	*Object-Relational Mapping*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UTFPR	Universidade Tecnológica Federal do Paraná
XHTML	*eXtensible HTML*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho que é composta pelas considerações iniciais com o contexto no qual se insere o sistema implementado como resultado deste trabalho, os objetivos e a justificativa do mesmo e a organização do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Uma fábrica de software pode ter escopos de atuação distintos. Para Fernandes e Teixeira (2004) uma fábrica de software pode abranger desde um projeto de software completo, até somente a realização do projeto físico ou a codificação de programas. Contudo, eles ressaltam que a produção fabril de software deve ser vista como um processo amplo, integrado, que envolve desde a arquitetura da solução até testes de aceitação, tendo agregados a implantação de processos, hardware, software e serviços, equipamentos de rede e comunicação e outros.

Contudo, independentemente da abrangência das atividades de ciclo de vida de software realizadas por uma fábrica de software é comum entre os diversos autores que a qualidade dos produtos produzidos por essas fábricas está relacionada à qualidade dos processos utilizados no ciclo de vida de software. Esses processos definem e orientam a realização das atividades por meio das quais são obtidos resultados que direta e indiretamente compõem os produtos da fábrica. Esses produtos abrangem software, realização de testes, elaboração de projetos e demais artefatos e serviços que são fornecidos por essas fábricas.

A disciplina de Oficina de Projeto e Desenvolvimento do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR), Câmpus Pato Branco visa simular uma fábrica de software, abrangendo da definição dos requisitos a implantação do sistema em ambiente de teste. Pretende-se que nessa disciplina os alunos desenvolvam um sistema de software de forma cooperativa e colaborativa. E que esse ciclo de vida seja amparado por modelos de qualidade.

Para que se possa mais facilmente simular um ambiente de empresa de desenvolvimento de software e visando trabalhar diversos conteúdos vistos durante o curso,

com ênfase em disciplinas da área de Engenharia de Software, o desenvolvimento terá como base processos. Esses processos foram definidos tendo como base modelos e normas de qualidade e são utilizados modelos e padronizações para a produção desses artefatos.

Embora o aplicativo computacional resultado deste trabalho tenha sido projetado para uma disciplina e curso específicos, o mesmo se aplica para o gerenciamento de fábricas de software que sejam definidas como um conjunto de processos integrados e que são definidos de acordo com modelos de qualidade. Quer essas fábricas estejam no contexto de disciplinas curriculares ou em atuação no mercado.

1.2 OBJETIVOS

O objetivo geral se refere ao resultado principal da realização deste trabalho que é a implementação de um aplicativo computacional. Os objetivos específicos complementam o objetivo geral, tanto em termos das tecnologias utilizadas como da finalidade da aplicação.

1.2.1 Objetivo Geral

Implementar um aplicativo *web* para gerenciamento das atividades realizadas em disciplinas de oficina de projeto e desenvolvimento de software.

1.2.2 Objetivos Específicos

- Propor uma solução para facilitar o gerenciamento do desenvolvimento de projetos em disciplinas de oficina de projeto e desenvolvimento de software.
- Possibilitar que disciplinas de oficina de projeto e desenvolvimento de software sejam conduzidas como uma fábrica de software, em termos de gerenciamento do ciclo do desenvolvimento de software.

- Oferecer a solução para implementar uma ferramenta que facilite o gerenciamento das equipes envolvidas no desenvolvimento colaborativo de projetos de software e as atividades das equipes que desenvolvem esses projetos.

1.3 JUSTIFICATIVA

A justificativa da proposta deste trabalho tem como base a necessidade de controle das atividades a serem realizadas pelas equipes de alunos da disciplina de projeto e desenvolvimento de software do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Esse controle é em termos de requisitos definidos para cada equipe e das atividades sendo realizadas. Contudo, ressalta-se que embora a modelagem seja realizada tendo como base essa disciplina, o sistema se aplicará para disciplinas com contextos e objetivos semelhantes e mesmo para fábricas de software.

Em termos da disciplina de Oficina de Projeto e Desenvolvimento de Software, cada turma de alunos (uma turma é composta por todos os alunos que cursam a disciplina em um determinado semestre) desenvolverá um mesmo projeto e esses alunos estarão organizados em equipes. Assim, é fundamental que o professor e mesmo os líderes (gerentes) das equipes possuam uma ferramenta específica de controle. É essa ferramenta que será desenvolvida como resultado deste trabalho.

Para que o projeto desenvolvido na disciplina de Oficina possa estar envolvido em um contexto de fábrica de software é necessário que sejam utilizados processos, modelos de artefatos, padronizações, que sejam seguidos procedimentos e que haja controles. Para atender a isso, esses processos serão definidos tendo como base modelos de qualidade.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a ideia do sistema, incluindo os objetivos e a justificativa.

No Capítulo 2 está o referencial teórico sobre fábricas de software, modelos de qualidade e processos. Esses conteúdos fundamentam a concepção conceitual de uma fábrica de software definida a partir de processos, sendo esses processos elaborados a partir de modelos de qualidade.

O Capítulo 3 contém o referencial teórico, baseado em *frameworks* para desenvolvimento de aplicações *web*. O sistema implementado como resultado deste trabalho é um aplicativo para *web* e foi desenvolvido com tecnologias que o caracterizam como de interface rica. Portanto, o referencial teórico está centrado nessas tecnologias.

No Capítulo 4 estão os materiais e o método utilizados. Os materiais se referem às ferramentas, linguagens e demais recursos necessários para implementar o sistema. O método contém as etapas necessárias para realizar o trabalho.

O Capítulo 5 contém a modelagem do sistema, exemplos de interface e as funcionalidades genéricas implementadas que visam facilitar a implementação do sistema.

No Capítulo 6 está a conclusão. E por fim estão as referências bibliográficas utilizadas na elaboração do trabalho.

2 FÁBRICA DE SOFTWARE DEFINIDA A PARTIR DE PROCESSOS

Este capítulo está centrado em conceitos e concepções de fábrica de software, processo de desenvolvimento de software e qualidade.

2.1 CONCEITOS DE FÁBRICA DE SOFTWARE

A produção de software é uma atividade essencialmente dependente de conhecimento e trabalho humanos. Ainda que existam muitas técnicas, tecnologias, ferramentas e linguagens que visam facilitar, agilizar e automatizar atividades de ciclo de vida de software e a produção de código - geradores de código, funcionalidades nativas (em ambientes de desenvolvimento, plataformas e outros) e facilidades de paradigmas e linguagens - a análise, o projeto e a implementação requerem a habilidade de pessoas. E é necessário que as pessoas que realizam essas atividades tenham conhecimento de linguagens e tecnologias de desenvolvimento de software, de análise e de projeto de sistemas, do domínio do negócio, das particularidades de cada projeto realizado e cada negócio automatizado, dentre outros.

Sob essa perspectiva, as atividades de desenvolvimento de software são muito distintas de uma produção fabril em série. Contudo, a associação da produção de software a um processo de fabricação em massa não decorre de itens obtidos em uma linha de produção, mas da forma organizada, baseada em processo e em requisitos de qualidade que define uma fábrica de bens tangíveis.

Em um escopo de uso de ferramentas para automatizar a produção de software, Greenfield e Short (2003) definem fábrica de software como uma linha de produto que permite configurar ferramentas extensíveis, processos e conteúdo, por meio de um modelo baseado em um esquema. O esquema é utilizado para automatizar o desenvolvimento e a manutenção das variações de um produto, pela adaptação, montagem e configuração de um *framework* baseado em componentes. Demir (2006) compara esse esquema a uma receita, na qual os ingredientes são as ferramentas para os processos e o modelo é o recipiente que contém os ingredientes. Nesse esquema, uma fábrica de software fundamenta-se amplamente em modelos e automação, que são os conceitos básicos do *Model Driven Development* (MDD), e ela é vista como uma metodologia de desenvolvimento de software focada no desenvolvimento de linhas de produtos.

Além do uso de ferramentas, uma fábrica de software pode ser entendida do ponto de vista de processos. Fernandes e Teixeira (2004) definem fábrica de software como um processo estruturado, controlado e melhorado continuamente. Esse processo considera a abordagem da engenharia industrial, é orientado para o atendimento a demandas de natureza distinta e visa à geração de produtos de software conforme requisitos documentados, da forma mais produtiva e econômica possível.

Um conceito mais amplo para fábrica de software, considerando-a como o ambiente para a realização de processos, é proveniente de Fabri et al. (2004). Esses autores conceituam fábrica de software como uma organização estruturada, voltada para a produção de software alicerçada na engenharia e com forte caracterização pela organização do trabalho e na capacidade de modularização de componentes e de escalabilidade produtiva.

Uma fábrica de software, vista como o ambiente de desenvolvimento de atividades relacionadas ao ciclo de vida software, pode destinar-se ao desenvolvimento de um projeto completo ou somente de atividades específicas, como, por exemplo, a produção de código ou a realização de testes.

Para Fernandes e Teixeira (2004), a produção fabril de software deve ser vista como um processo amplo e integrado, que abrange da arquitetura da solução até os testes de aceitação. Por outro lado, a existência de ambientes e de linguagens de programação configuráveis que visam agilizar a produção de software têm contribuído para vincular o conceito de fábrica de software às linguagens de domínio específico e às linhas de produto (GREENFIELD; SHORT, 2003; DEMIR, 2006; FRANKEL, 2005).

Considerando a relevância da qualidade de software como produto e que a qualidade do produto está relacionada à qualidade do processo, a definição de uma fábrica de software a partir de processos pode fornecer uma estrutura mais propícia ao desenvolvimento de software com qualidade. Processos padrão, definidos de acordo com normas e modelos de qualidade, podem ser especializados para produzir produtos de software de domínio específico ou para adequar-se às características de cada fábrica.

O uso de processos adequadamente definidos no desenvolvimento de software é considerado um aspecto relevante para desenvolver software que atenda aos requisitos do usuário e que seja desenvolvido de acordo com o planejamento e o orçamento (FUGGETTA, 2000; BERTOLLO; FALBO, 2003). Um processo adequadamente definido determina as atividades que devem ser realizadas, quem as realiza, os recursos utilizados para realizá-las e os resultados produzidos, além de possibilitar a definição de políticas e controles de qualidade. Para Fiorini, Von Staa e Baptista (1998) conhecer os processos significa conhecer

como os produtos são planejados e produzidos. E se os processos estão adequadamente documentados é possível conhecê-los, bem como os seus resultados e os controles definidos antes mesmo da sua execução.

Apesar da existência de modelos e normas de qualidade para a definição de processos de software, os processos precisam considerar as particularidades de desenvolvimento e interesses e necessidades dos usuários do software. Dentre essas particularidades estão a adequação às tecnologias utilizadas, ao tipo de software, a capacitação e as características da equipe, da organização e do projeto (ROCHA; MALDONADO; WEBER, 2001). Devido à quantidade de variáveis envolvidas na realização de um projeto de software, pode ser necessário definir processos específicos para cada projeto de software que a fábrica de software desenvolve.

Contudo, embora os projetos de software geralmente tenham características específicas é possível estabelecer um conjunto de processos comuns, definidos como padrão que podem ser instanciados de acordo com as especificidades de cada projeto. Esses processos constituem processos padrão de desenvolvimento de software (BERTOLLO; FALBO, 2003). Esses processos são instanciados considerando as características e as particularidades da organização, da equipe, das tecnologias e ferramentas utilizadas, dos requisitos do sistema, dentre outros.

2.2 FÁBRICA DE SOFTWARE BASEADA EM PROCESSOS

Um processo é definido como uma rede de atividades relacionadas entre si, com critérios para indicar o seu início e fim e com informações sobre as atividades realizadas, os envolvidos e as tecnologias de informação e os dados vinculados (WFMC-TC 1011, 1996).

De acordo com a NBR ISO/IEC 12207 (ASSOCIAÇÃO..., 2009), processo é um conjunto de atividades inter-relacionadas que transformam entradas em saídas. Weber et al. (2005) apresentam complementações e definem um processo de software como um conjunto de atividades, métodos, práticas e transformações empregadas para desenvolver e manter software e produtos associados. Bertollo e Falbo (2003) incluem, ainda, a definição de um modelo de ciclo de vida, os artefatos (insumos e produtos) requeridos e produzidos para cada uma das atividades do processo, os procedimentos (definidos como métodos, técnicas, normas e roteiros, dentre outros) a serem adotados e os recursos necessários (pessoas, equipamentos, ferramentas, tecnologias, ambientes, por exemplo) para a realização das atividades.

Mangan e Sadiq (2002) definem quatro componentes primários para um processo que são: objetos, tarefas, executores e restrições. Objetos se referem ao que é manipulado pelas tarefas. Executores realizam tarefas produzindo e utilizando objetos. Restrições são, por exemplo, as políticas que restringem as ações dos executores das atividades.

Taylor (2003) denomina organização, processo e recurso como os elementos de negócio básicos de representação. A partir desses elementos, a organização é estruturada como um conjunto de classes com responsabilidades.

Anaya e Ortiz (2005) sustentam a ideia de ter processo como base para o gerenciamento. A existência de processos bem definidos pode contribuir para a flexibilidade e a melhoria contínua em uma fábrica de software. A flexibilidade é importante para que a fábrica possa desenvolver projetos distintos (como caracteristicamente são os projetos de software), adaptar-se às mudanças das tecnologias e as necessidades dos clientes.

No contexto deste trabalho, fábrica de software é definida como um ambiente de desenvolvimento de software organizada por processos que são definidos a partir de modelos de qualidade. Esses processos são compostos por atividades, realizadas por atores que instanciam papéis e utilizam recursos para produzir artefatos de software e estão sujeitos a políticas e métricas de qualidade.

O uso de processos para definir uma fábrica de software a aproxima, de certa forma, das concepções de uma linha de manufatura industrial que é flexível, no sentido de ser composta por processos que podem ser adaptados para o desenvolvimento de projetos de software distintos. Assim, o conceito fábrica de software utilizado neste trabalho sugere uma maneira de organizar a realização das atividades por meio de processos que se flexibilizam e se adaptam às necessidades dos projetos sendo realizados e das condições da própria fábrica. Essas necessidades podem ser internas (composição, conhecimento e necessidade de aprendizados da equipe, por exemplo) ou externas (como inovações tecnológicas e necessidades e interesses dos clientes e usuários).

2.3 QUALIDADE APLICADA A PROCESSOS DE SOFTWARE

Dentre as normas e os modelos de qualidade utilizados na área de software estão: ISO 9000 (INTERNATIONAL..., 2000), NBR ISO/IEC 12207 (ASSOCIAÇÃO..., 2009), ISO/IEC 15504 (INTERNATIONAL..., 1999), CMMI (SOFTWARE..., 2006) e MPS.BR (SOFTEX, 2007), sucintamente apresentados a seguir.

As normas NBR ISO 9000:2000 (INTERNATIONAL..., 2000) foram desenvolvidas para apoiar organizações, independentemente do tipo de atividade que exercem ou do seu porte, na implementação e na operação de sistemas de gestão da qualidade eficazes.

A norma NBR ISO/IEC 12207 (ASSOCIAÇÃO..., 2009) visa fornecer uma estrutura comum para que adquirentes, fornecedores, desenvolvedores, mantenedores, operadores, gerentes e técnicos envolvidos com o desenvolvimento de software. Essa estrutura é representada por uma linguagem que é estabelecida na forma de processos. Os processos da norma, executados durante o projeto de software, visam prover qualidade ao produto e ao processo.

A norma internacional ISO/IEC 15504 (INTERNATIONAL..., 1999) estabelece uma estrutura para a avaliação de processos. Essa estrutura pode ser usada por organizações envolvidas com planejamento, gerenciamento, monitoramento, controle e melhoria de aquisição, fornecimento, desenvolvimento, operação, evolução e suporte de software.

O CMMI (SOFTWARE..., 2006) provê uma estrutura que auxilia as organizações na avaliação de sua maturidade ou capacidade em determinada área de processo e no estabelecimento de prioridades para prover melhorias e implementá-las. Um dos objetivos do CMMI é fornecer direcionamentos para melhorar os processos de uma organização e sua capacidade de gerenciar o desenvolvimento, a aquisição e a manutenção de produtos e serviços.

O modelo de qualidade de processo Melhoria de Processo do Software Brasileiro, MPS.BR (SOFTEX, 2007), é destinado às empresas de desenvolvimento de software de pequeno e médio porte. Esse modelo é baseado nas normas ISO/IEC 12207 e ISO/IEC 15504 e é compatível com o CMMI.

3 FRAMEWORKS E BIBLIOTECAS PARA DESENVOLVIMENTO PARA WEB

Este capítulo apresenta o referencial teórico do trabalho que fundamenta as tecnologias e as ferramentas utilizadas no desenvolvimento do aplicativo resultado deste trabalho. Esse referencial se refere ao desenvolvimento de aplicações *web* e *frameworks*. O aplicativo a ser desenvolvido é um sistema para *web* com interface rica e será desenvolvido utilizando *frameworks* e biblioteca para implementar esse tipo de interface.

3.1 DESENVOLVIMENTO DE APLICAÇÕES WEB

Os aplicativos *web* são baseados no modelo cliente/servidor. Os clientes fazem solicitações e recebem respostas de servidores. Os servidores atendem essas solicitações e disponibilizam informações para os clientes. A comunicação entre servidor e cliente é realizada por meio do protocolo *HyperText Transfer Protocol* (HTTP) que é baseado no *Transmission Control Protocol/Internet Protocol* (TCP/IP) (CELEPAR, 2009). De maneira sucinta, essa é a forma básica de atuação dos aplicativos baseados na Internet, sejam sites simples para disponibilizar informações ou sistemas corporativos complexos, distribuídos e que manipulam bases de dados com *terabytes* de dados armazenados.

A *web* como o ambiente gráfico da Internet, caracterizada como um serviço Internet, foi inicialmente utilizada para disponibilizar páginas de texto vinculadas entre si por meio de ligações denominadas *hiperlinks*. Atualmente são muitas as empresas e instituições que possuem seus sistemas corporativos que executam por meio de serviços e da infraestrutura da Internet, seja no como *intranet* ou *extranet*. Com a expansão do uso da infraestrutura da Internet para executar aplicativos, que estavam restritos aos ambientes *desktop* e redes corporativas, recursos de interação existentes nesses aplicativos passaram a ser solicitados pelos usuários. A *HyperText Markup Language* (HTML) como a linguagem para a definição da interface desses sistemas *web* começou a apresentar limitações. Os usuários de aplicações *desktop* estavam acostumados a recursos de interação com muitos mais recursos do que os oferecidos pelas páginas *web* baseadas em hipertexto e formulários com componentes simples.

Para atender a essa necessidade, muitos recursos como bibliotecas de componentes gráficos, linguagens de *scripts* que executam no lado cliente e *frameworks* passaram a ser desenvolvidos como forma de melhorar e facilitar a interatividade das aplicações *web*.

3.2 FRAMEWORKS PARA DESENVOLVIMENTO WEB

Um *framework* é um conjunto de componentes integrados entre si que colaboram para produzir uma arquitetura reusável para uma família de aplicações (OKANOVIEC, 2011). As funcionalidades providas pelos *frameworks* devem ser expandidas de forma a atender as características específicas da aplicação que está sendo desenvolvida.

Buschmann et al. (1996) definem *framework* como um software parcialmente completo que é projetado para ser instanciado. Um *framework* também é definido como sendo uma aplicação para funcionalidades específicas, composta por uma estrutura estática e outra dinâmica, desenvolvidas para resolver um conjunto restrito de problemas (FAYAD, 2000).

De forma prática e voltada para a implementação de sistemas computacionais, *frameworks* são vistos como estruturas de código, nas quais um projeto de software, seja *web* ou não, pode ser criado e desenvolvido. *Frameworks* podem incluir programas de suporte, bibliotecas de código, linguagens de *script* ou qualquer outro tipo de software para auxiliar no desenvolvimento (ROGERIO, 2008).

Algumas funcionalidades são consideradas desejáveis em *frameworks* caracterizados como voltados ao desenvolvimento rápido de aplicações. Dentre elas, estão a implementação automática de funções como CRUD (*Create, Retrieve, Update and Delete*), ORM (*Object-Relational Mapping*) e de padrões como o MVC (*Model-View-Controller*) (PEREIRA; COGO; CHARÃO, 2009).

Os *frameworks* podem ser classificados em de aplicação e de componentes (BARRETO JUNIOR, 2006). Para Barreto Junior (2006) um *framework* de aplicação orientados a objetos define uma solução inacabada que gera uma família de aplicações e um *framework* de componentes estabelece um contrato para conectar componentes.

Para Fayad et al (1999) *frameworks* de aplicação são classificados quanto ao seu escopo em *frameworks* de: infra-estrutura de sistemas, integração de *middleware* e aplicações corporativas.

Um *framework* de componentes é uma entidade de software que provê suporte a componentes que seguem um determinando modelo e possibilitam que instâncias desses

componentes sejam conectadas no *framework* (SZYPERSKI, 1997). O *framework* estabelece as condições necessárias para que um componente possa ser executado e define a forma de interação entre as instâncias desses componentes.

A importância atual dos *frameworks* de componentes pode ser verificada pela publicação de Buchner, Böttcher e Storch (2012) que compuseram uma lista com 110 *frameworks.*

Na seção a seguir é apresentado sobre o *framework* JavaServer Faces porque o mesmo é utilizado para implementar o sistema modelado como resultado deste trabalho.

3.2.1 JavaServer Faces

JavaServer Faces (JSF) é um dos principais *frameworks* Java para *web* (SENGER; MAGALHÃES, 2013), o seu desenvolvimento é baseado em componentes e permite a extensão de componentes existentes e a criação de novos componentes por meio de *eXtensible HTML* (XHTML). JSF implementa um modelo de eventos simulando um aplicativo *desktop*, mas adota o padrão de projetos MVC e isso permite encapsular a infraestrutura do ambiente *web*. JSF possui suporte nativo para Ajax e *annotations*.

A tecnologia JSF inclui (ORACLE, 2013):

a) Um conjunto de componentes para representar a interface gráfica com o usuário e gerenciar o seu estado, manipular eventos e realizar a validação de entrada, definir navegação nas páginas e suportar internacionalização e acessibilidade.

b) Uma biblioteca de *tags* personalizadas *JavaServer Pages* (JSP) para expressar a interface JSF dentro de páginas JSP.

A extensão dos componentes é realizada por meio de bibliotecas, dentre as quais estão (SENGER; MAGALHÃES, 2013): RichFaces (RICHFACES, 2013), IceFaces (ICEFACES, 2013) e PrimeFaces (PRIMEFACES, 2013), MyFaces Tomahawk, Trindad, PrettyFaces, Woodstock e WebGalileo Faces.

A Figura 1 apresenta uma visão geral do funcionamento de uma aplicação com JSF. Cada requisição JSF que renderiza uma JSP envolve uma árvore de componentes, também chamada de visão e ocorre por meio de um ciclo de vida de processamento da requisição que é realizada por fases. As fases padrão do ciclo de vida de processamento da requisição iniciam com a construção da visão de restauração, em seguida valores de requisição são aplicados,

validações são processadas, valores de modelos são atualizados e a aplicação é invocada (IBM, 2013).

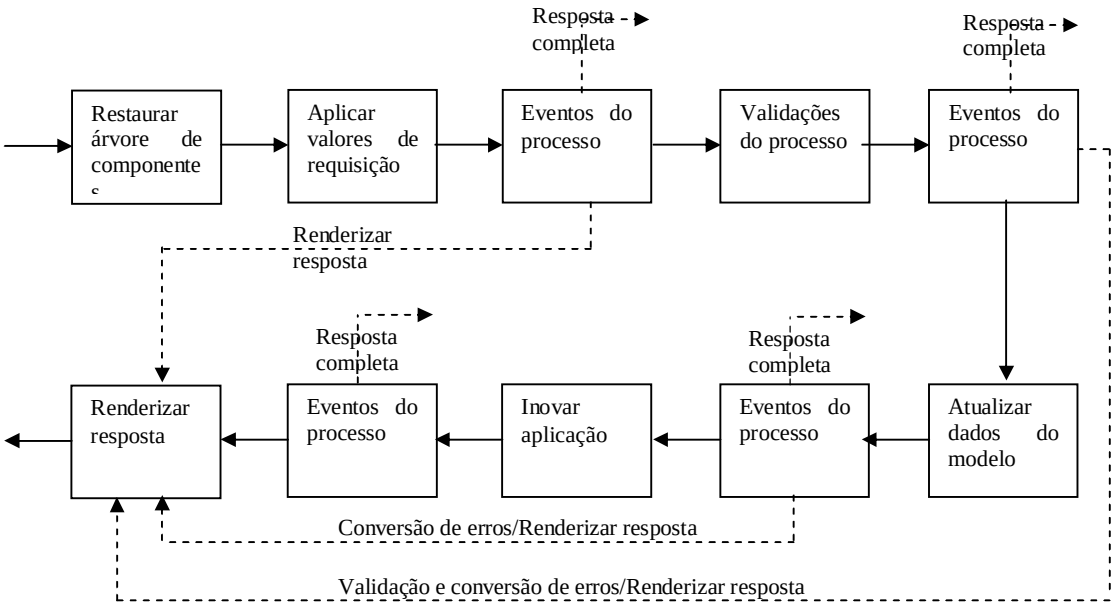

Figura 1 – Ciclo de vida de uma aplicação com JSF
Fonte: traduzido de IBM (2013).

De acordo com a Figura 1:

a) Restaurar árvore de componentes – a árvore de componentes JSF é usada para construir e manter os estados e os eventos para a página. A árvore é construída uma vez por sessão e reusada quando os usuários retornam à página. Ao final desta fase, a propriedade raiz da instância do FacesContext para a requisição atual reflete as configurações salvas para a visão gerada pelo Faces Response anterior, se há algum.

b) Aplicar valores de requisição – o propósito desta fase do ciclo de vida do processamento de uma requisição visa oportunizar que cada componente atualize seu valor atual usando a informação incluída na requisição atual. Dentre esses valores estão: parâmetros, cabeçalhos e *cookies*.

c) Eventos do processo – durante várias fases do ciclo de vida de processamento, eventos podem ser colocados em uma fila de eventos. Uma *flag* booleana é retornada indicando se o evento foi completamente realizado e se a implementação JSF pode removê-lo da lista.

d) Validações do processo – como parte da criação da visão para a requisição, zero ou mais instâncias do validador podem ser registradas para cada componente. Em

adição, as próprias classes de componentes podem implementar lógica de validação em seus métodos validate(). Ao final da fase, todas as validações configuradas são completadas. Validações que falham causam mensagens para serem enfileiradas por meio de chamadas para o método addMessage() da instância do FacesContext para a requisição atual e a propriedade válida para os componentes correspondentes são marcadas como para falso.

e) Atualizar dados do modelo – nesta fase o ciclo de vida do processo de requisição é alcançado, isso significa que as requisições estão sintática e semanticamente validadas. Os valores locais de cada componente na árvore de componentes é o modelo de dados da aplicação são atualizados como uma preparação para realizar qualquer evento da aplicação que está na fila.

f) Invocar aplicação – se a visão de uma requisição atual foi reconstruída a partir do estado da informação salvo por uma requisição anterior, a implementação JSF terá certeza que o ActionListner retornado pela chamada de getActionListener no objeto da aplicação será registrado com todos os componentes da interface gráfica na árvore de componentes.

Renderizar resposta – esta fase realiza duas atividades ao mesmo tempo: faz com que a resposta seja renderizada (apresentada) para o cliente e faz que o estado da resposta seja salvo para processamento ou requisições posteriores.

4 MATERIAIS E MÉTODO

Este capítulo contém as ferramentas, as tecnologias e o método utilizados para a modelagem e a implementação do sistema. Os materiais incluem linguagem de programação, banco de dados, interface e tecnologias de desenvolvimento e editores utilizados para a modelagem do sistema. O método se refere às principais atividades realizadas para obter o resultado do trabalho.

4.1 MATERIAIS

Para a modelagem e a implementação do sistema foram utilizadas as ferramentas e as tecnologias apresentadas no Quadro 1. Nesse quadro também é apresentada uma breve descrição de uso, indicando a aplicação de cada uma das ferramentas e tecnologias utilizadas no desenvolvimento do trabalho.

Tecnologia /Ferramenta	**Versão**	**Site**	**Descrição de uso**
Astah Community	6.2.1	http://astah.net/editions/community	Modelagem do diagrama com a visão geral do sistema e diagrama de casos de uso
Case Studio 2	2.25.0	http://www.casestudio.com	Modelagem do diagrama de entidades e relacionamentos do banco de dados
Netbeans	7.4	https://netbeans.org/	Ambiente de Desenvolvimento
MySQL Cluster	5.6	http://www.mysql.com/	Sistema Gerenciador de Banco de Dados
MySQL Workbench	6.0	http://www.mysql.com/	Modelagem do diagrama de entidades e relacionamentos do banco de dados
Glassfish	4.0	https://glassfish.java.net/	Servidor *web* de aplicações
Java EE	7	http://www.oracle.com/technetwork/java/javaee/overview/index.html	Especificação Java para desenvolvimento de aplicações *web*
CDI (WeldFly)	2.1	http://weld.cdi-spec.org/	Injeção de contextos e dependências de classes, propriedades e métodos da aplicação
JPA/JTA (Hibernate)	4.3	http://hibernate.org/	*Framework* para persistência de dados e controle de transações
JavaServer Face	2.2	http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html https://javaserverfaces.java.net/	*Framework* para desenvolvimento da interface
PrimeFaces	4.0	http://primefaces.org/	Biblioteca de componentes para desenvolvimento da interface

Quadro 1 – Ferramentas e tecnologias

4.2 MÉTODO

A organização das atividades para o desenvolvimento do sistema para gerenciamento da disciplina de Oficina de Projeto e Desenvolvimento de Software foi realizada a partir do modelo sequencial linear como descrito em Pressman (2008) e do processo unificado (BLAHA et al., 2006). O modelo sequencial linear forneceu a denominação das etapas e as suas atividades. O processo unificado foi utilizado na definição dos ciclos iterativos de desenvolvimento.

A modelagem do sistema foi realizada como trabalho de estágio do aluno Patrick Cecato Albani. O estudo das tecnologias, com a definição de funcionalidades básicas que pudessem ser reusadas e agilizar a implementação, foi realizado como trabalho de estágio do aluno Alexandre de Carli. Portanto. Portanto,, no Quadro 2 essas atividades não constam.

O Quadro 2 apresenta os processos (fluxos de trabalho) e as iterações planejadas e realizadas.

Iterações / Processos	1ª iteração	2ª iteração	3ª iteração
Requisitos	Revisão dos requisitos.		
Análise e projeto	Ajustes na modelagem. Revisão e ajustes no diagrama de entidades e relacionamentos.		
Implementação	Implementação dos cadastros básicos a partir do GenericBean implementado.	Implementação dos demais cadastros.	Implementação das funcionalidades de gerenciamento e dos relatórios.
Testes	Teste funcional de cadastros básicos.	De código, realizados pelo autor deste trabalho.	De código, realizados pelo autor deste trabalho. Das funcionalidades e de interação com o sistema pela orientadora.

Quadro 2 – Iterações definidas

A seguir estão descritas as etapas definidas para o desenvolvimento do aplicativo e as principais atividades de cada uma dessas etapas.

a) Requisitos

A revisão teve como finalidade ajustar os requisitos de sistema definidos com base em mudanças indicadas pelas professoras Beatriz T. Borsoi, Eliane D. B. Fávero e Lucilia Y. Araki. Essas professoras forneceram a visão geral do sistema, com a indicação dos requisitos essenciais.

b) Modelagem dos requisitos

As tabelas para o banco de dados foram ajustadas e complementadas em decorrência das mudanças de requisitos realizadas.

c) Implementação

A implementação dos cadastros teve como base o GenericBean criado. Em seguida foram implementadas funcionalidades mais complexas relacionadas ao gerenciamento da fábrica. Por fim foram implementados os relatórios.

e) Testes

Os testes unitários (de código) foram realizados pelos autores deste trabalho e de funcionalidades e de interação com o sistema pela professora orientadora.

5 RESULTADOS

Este capítulo apresenta o resultado da realização deste trabalho que é a modelagem de um sistema para gerenciamento de disciplinas de oficina de projeto e desenvolvimento de sistemas como fábricas de software. É, também, apresentada a implementação de funcionalidades básicas desenvolvidas com o objetivo de agilizar a implementação.

5.1 ESCOPO DO SISTEMA

A concepção de fábrica de software para a disciplina de Oficina de Projeto de Desenvolvimento abrange o ciclo de vida de software nos processos relacionados à definição e modelagem dos requisitos, à implementação, aos testes e à instalação do sistema. A esses processos estarão vinculados modelos e procedimentos de garantia da qualidade, gerência de configuração e outros pertinentes ao nível F do MPS-BR (SOFTEX, 2011).

O aplicativo computacional desenvolvido se destina ao gerenciamento de fábricas de software definidas como um conjunto de processos integrados e elaborados de acordo com modelos de qualidade. A sua utilização em disciplinas acadêmicas visa facilitar o trabalho do professor no gerenciamento das atividades e das equipes de projeto (os alunos da disciplina).

5.2 MODELAGEM DO SISTEMA

A Figura 2 apresenta a visão geral do sistema com os principais conceitos envolvidos no gerenciamento de uma disciplina de laboratório de desenvolvimento de software, conforme o contexto e escopo considerados neste trabalho.

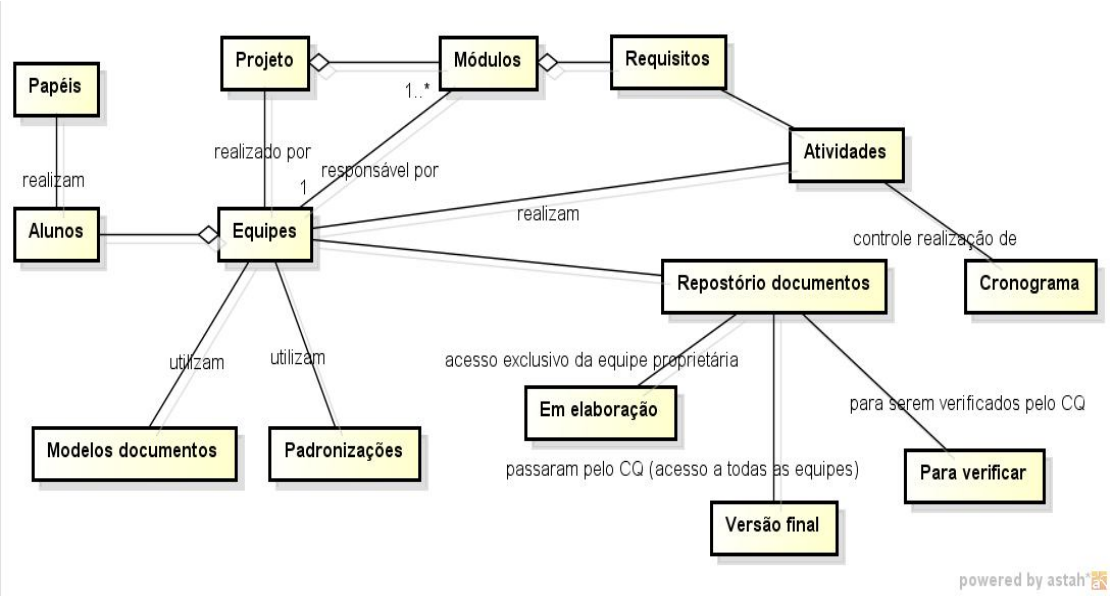

Figura 2 – Visão geral do sistema

O Quadro 3 apresenta os requisitos funcionais identificados para o sistema.

Identificação	**Nome**	**Descrição**
RF01	Manter equipes	Incluir, excluir, alterar, consultar e compor equipes pertencentes ao projeto. Uma equipe é composta por um conjunto de alunos. As equipes são responsáveis pelo desenvolvimento dos módulos do sistema e por outras atividades como o controle de qualidade.
RF02	Manter alunos	Incluir, excluir, alterar e consultar alunos que compõem as equipes que estão realizando um projeto.
RF03	Manter papéis	Incluir, excluir, alterar e consultar papéis realizados pelos membros de uma equipe no desenvolvimento de um projeto. Um aluno pode realizar mais de um papel simultaneamente.
RF04	Manter projetos	Incluir, excluir, alterar projeto. Um projeto representa o sistema de software que é modelado e implementado pelas equipes.
RF05	Manter módulos	Incluir, excluir, alterar módulos. Um módulo é um agrupamento de requisitos do sistema. Um módulo determina o que é de responsabilidade de cada equipe modelar, implementar e documentar.
RF06	Manter requisitos	Incluir, excluir, alterar requisitos. É a listagem de requisitos definidos para o sistema. Esses requisitos podem ser de alto nível e definidos apenas para orientar a distribuição das funcionalidades para as equipes. Cada equipe trabalha os requisitos no sentido de dividi-los ou complementá-los.
RF07	Manter atividades	Incluir, excluir, alterar e consultar atividades. As atividades se referem à modelagem e implementação das funcionalidades do módulo e a atividades inter-módulos como as de controle de qualidade e planejamento do projeto.
RF08	Estabelecer vínculos	Apresentar os vínculos como uma matriz. Em princípio os vínculos são entre casos de uso, casos de uso e requisitos,

		casos de uso e classes, casos de uso e tabelas. O usuário poderá selecionar o tipo de vínculo que quer visualizar.

Quadro 3 – Requisitos funcionais estabelecidos para o sistema

A listagem do Quadro 4 apresenta os requisitos não-funcionais identificados para o sistema. Esses requisitos explicitam regras de negócio, restrições ao sistema de acesso, por exemplo, requisitos de qualidade, desempenho, segurança e outros.

Identificação	Nome	Descrição
RNF01	Acesso ao repositório de documentos	A área do repositório exclusiva de cada equipe poderá ser acessada somente pela respectiva equipe.
RFN02	Composição de equipes	A composição de equipes é realizada pelo administrador do sistema que pode ser o professor da disciplina.
RNF03	Alterações realizadas	O sistema deve avisar de alterações realizadas nos requisitos, classes, tabelas e casos de uso. Sempre que uma alteração é documentada o sistema deve emitir um aviso.

Quadro 4 – Requisitos não funcionais estabelecidos para o sistema

A partir dos requisitos foram definidos os casos de uso apresentados na Figura 2.

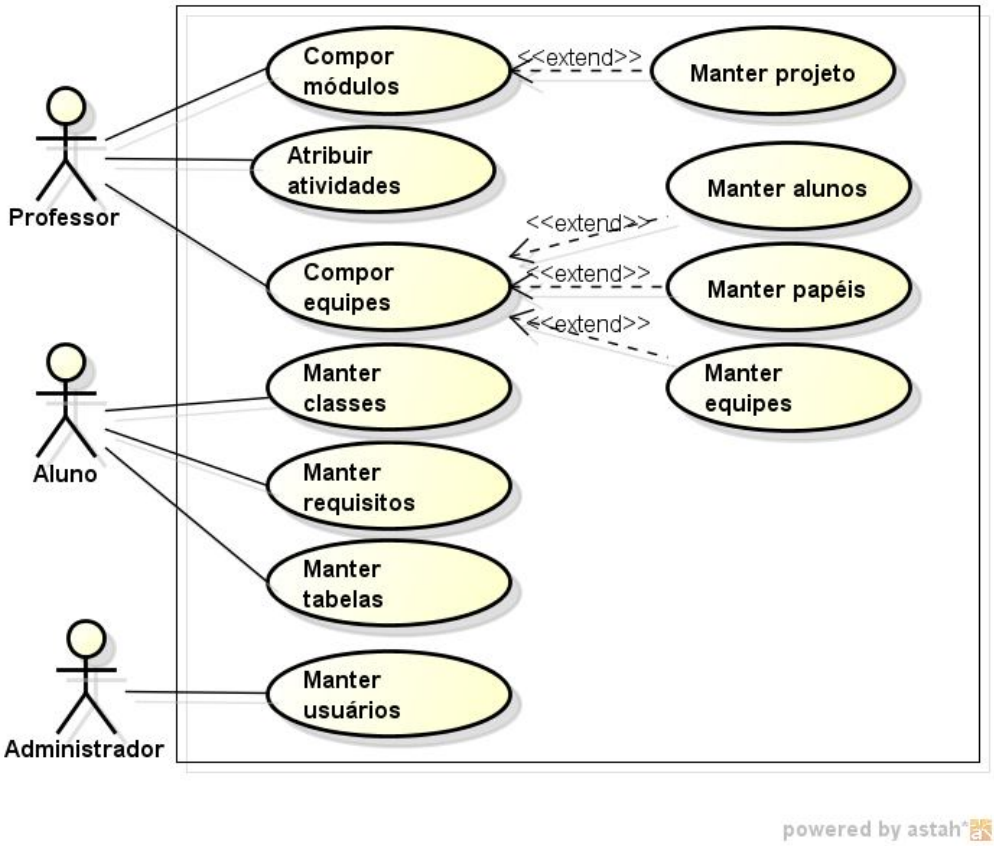

Figura 3 – Diagrama de casos de uso do sistema

No Quadro 5 está documentado um caso de uso de cadastro. Esses casos de uso são identificados com o estereótipo CRUD na Figura 6. O caso de uso documentado é o de "Manter projeto" e é utilizado para exemplificar como são descritos os casos de uso de cadastro. Todos os casos de uso seguem o mesmo padrão, considerando que pode ou não haver necessidade de dados provenientes de outros cadastros.

1 Identificador do caso de uso:
Manter projeto.
Descrição:
Cadastro de projetos aos quais estarão atividades vinculadas.
Evento Iniciador:
O usuário solicita a inclusão de um projeto no sistema.
Atores:
Professor, Administrador
Pré-condição:
O tipo de projeto deve estar cadastrado.
Sequência de Eventos:
1. Ator Professor/Administrador acessa a tela para cadastro de um novo projeto e inclui as informações necessárias. O tipo de projeto, um dos campos de entrada, deve estar cadastrado e é escolhido a partir de uma listagem apresentada.
2. O sistema insere os dados no banco de dados, verificando se o nome do projeto está descrito e informa o usuário que o referido projeto foi incluído.
Pós-Condição:
Projeto inserido no banco de dados.
Extensões:
Cadastrar tipo de projeto.

Nome do fluxo alternativo (extensão)	Descrição
1.1 Cadastro de tipo de projeto.	1.1 O ator professor ou administrador acessa a tela do sistema para cadastrar tipo de projeto pretendido e inclui as informações solicitadas. 1.2 Sistema inclui informações no banco de dados.

Quadro 5 – Caso de uso de cadastro

O Quadro 6 apresenta o caso de uso compor equipes.

2 Identificador do caso de uso:
Compor equipes
Descrição:
Cada equipe é composta por um grupo de alunos. Cada aluno realiza um ou mais papéis na equipe.
Evento Iniciador:
Tela para composição de equipes aberta.
Atores:
Professor
Pré-condição:
Alunos cadastrados.

Papéis cadastrados.

Sequência de Eventos:

1. Ator professor informa/seleciona identificação da equipe.
2. Ator professor selecionado o projeto para a equipe. A partir da seleção do projeto são disponibilizados para escolha os módulos do referido projeto. Dentre os módulos apresentados o professor seleciona um para vincular à equipe.
3. Ator professor vincula alunos à equipe.
4. Ator professor solicita cadastro da equipe.
5. Sistema cadastra equipe e informa que a operação foi realizada

Pós-Condição:

Equipe com membros vinculados.

Extensões:

Não há

Quadro 6 – Caso de uso compor equipes

No Quadro 7 está a descrição do caso de uso atribuir atividades às equipes.

2 Identificador do caso de uso:

Atribuir atividades às equipes.

Descrição:

As atividades devem ser atribuídas às equipes.

Evento Iniciador:

Tela para atribuição de atividades.

Atores:

Professor

Pré-condição:

Equipes cadastradas.
Atividades cadastradas.

Sequência de Eventos:

1. Ator professor seleciona a equipe.
2. Ator professor atribui atividade para a equipe.
5. Sistema cadastra atribuição e informa que a operação foi realizada

Pós-Condição:

Atividade atribuída para equipe.

Extensões:

Não há

Quadro 7 – Caso de uso atribuir atividades às equipes

A Figura 4 apresenta o diagrama de entidades e relacionamentos do banco de dados. Esse diagrama representa por meio de tabelas o contexto da Figura 1. Nesse diagrama estão entidades relacionadas a alunos e professores porque a instanciação do contexto apresentado na Figura 2 está sendo realizada para a disciplina de Oficina de Projeto e Desenvolvimento de Software do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Contudo, ressalta-se que o aplicativo pode ser utilizado para gerenciar fábricas de software organizadas por processos que são definidos de acordo com modelos de qualidade.

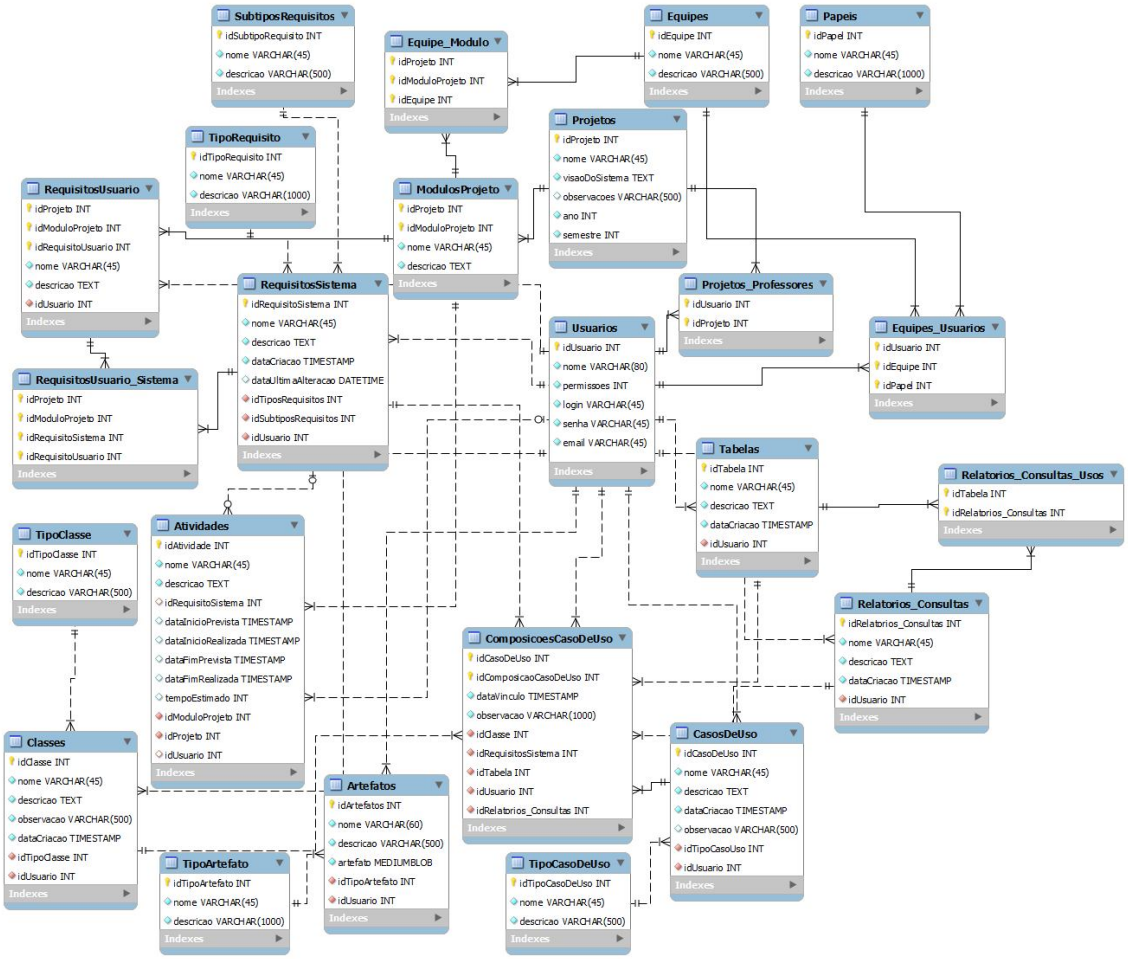

Figura 4 – Digrama de entidades e relacionamentos

Tabela Usuarios (Quadro 3) – armazena todos os usuários do sistema, incluindo alunos, professores e administrador.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idUsuario	Numérico	Sim	Não	
nome	Texto	Não	Não	
permissoes	Numérico	Não	Não	
login	Texto	Não	Não	
senha	Texto	Não	Não	Criptografada
email	Texto	Não	Não	

Quadro 8 – Campos da tabela Usuarios

Tabela Papeis (Quadro 4) – são os papéis que definem o conjunto de atividades que os respectivos atores irão realizar.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idPapel	Numérico	Sim	Não	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 9 – Campos da tabela Papeis

Tabela Equipes (Quadro 5) – as equipes que irão se responsabilizar pela realização das atividades que compõem projeto. As equipes serão responsáveis pelos módulos do projeto.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idEquipe	Numérico	Sim	Não	
Nome	Texto	Não	Não	
Descricao	Texto	Não	Não	

Quadro 10 – Campos da tabela Equipes

Tabela Equipes_Usuarios (Quadro 6) – armazena os relacionamento para instanciar os atores desempenhando determinado papel e participando de uma equipe.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
IdUsuario	Numérico	Não	Sim	Da tabela Equipes
IdEquipe	Numérico	Não	Sim	Da tabela Alunos
IdPapel	Numérico	Não	Sim	Da tabela Papeis

Quadro 11 – Campos da tabela Equipes_Usuarios

Tabela Projetos (Quadro 7) – são os dados do projeto a ser desenvolvido.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idProjeto	Numérico	Sim	Não	
nome	Texto	Não	Não	
visaoSistema	Texto	Não	Não	
observacoes	Texto	Não	Não	
ano	Data (ano)	Não	Não	
semestre	Numérico	Não	Não	

Quadro 12 – Campos da tabela Projetos

Tabela Projetos_Professores (Quadro 8) – contém o relacionamento do projeto com os usuários professores que serão os responsáveis pelo monitoramento e avaliação do trabalho realizado.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idProjeto	Numérico	Sim	Sim	Da tabela Projetos
idProfessor	Numérico	Sim	Sim	Da tabela Professores

Quadro 13 – Campos da tabela Projetos_Professores

Tabela Modulos_Projeto (Quadro 9) – Divisão do projeto em pacotes de atividades que serão desempenhadas por uma equipe.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idModuloProjeto	Numérico	Sim	Não	
idProjeto	Numérico	Não	Sim	Da tabela Projetos
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 14 – Campos da tabela Modulos_Projetos

Tabela Equipe_Modulo (Quadro 10) – Relacionamento da equipe com seus respectivos módulos que a mesma desenvolve.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idProjeto	Numérico	Sim	Não	
idModuloProjeto	Numérico	Sim	Sim	Da tabela Modulos_projeto
IdEquipe	Numérico	Sim	Sim	Da tabela Equipes

Quadro 15 – Campos da tabela Equipes_Modulos

Tabela TipoRequisito (Quadro 11) – Classificação a ser utilizada para os requisitos.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idTipoRequisito	Numérico	Sim	Não	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 16 – Campos da tabela Tipos_Requisitos

Tabela SubtiposRequisitos (Quadro 12) – para armazenar dados para a categorização de subtipos de requisitos.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idSubTipoRequisito	Numérico	Sim	Não	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 17 – Campos da tabela SubtiposRequisitos

Tabela Requisitos_Sistema (Quadro 13) – solicitação técnica de mudança e/ou criação de certa funcionalidade ou característica do sistema que está sendo desenvolvido.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idRequisitoSistema	Numérico	Sim	Não	
idUsuario	Numérico	Não	Sim	Da tabela Alunos
idTipoRequisito	Numérico	Não	Sim	Da tabela TipoRequisitos
idSubTipoRequisito	Numérico	Não	Sim	Da tabela SubTipoRequisitos
nome	Texto	Não	Não	
descricao	Texto	Não	Não	
dataCriacao	Data	Não	Não	
dataUltimaAlteracao	Data	Não	Não	

Quadro 18 – Campos da tabela Requisitos_Sistema

Tabela Requisitos_Usuario (Quadro 14) – requisitos definidos no ponto de vista do usuário. Esses requisitos representam os interesses e necessidades do usuário, bem como políticas aplicáveis ao sistema. Esses requisitos serão posteriormente transformados em requisitos do sistema, no sentido de fornecer a base para a definição dos requisitos do sistema.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idRequisitosUsuario	Numérico	Sim	Não	
idUsuario	Numérico	Não	Sim	Da tabela Uusarios
idProjeto	Numérico	Não	Sim	Da tabela RequisitosUsuario
idModuloProjeto	Numérico	Não	Sim	Da tabela ModulosProjeto
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 19 – Campos da tabela Requisitos_Usuario

Tabela RequisitosUsuario_Sistema (Quadro 15) – relacionamento dos requisitos do sistema com o do usuário, proporcionando então a solução de várias solicitações do usuário

por uma do sistema ou uma solicitação do sistema sendo detalhadas por vários requisitos do sistema ou relação de muitos para muitos.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idRequisitoUsuario	Numérico	Sim	Não	
idRequisitoSistema	Numérico	Não	Sim	Da tabela RequisitosSistema
idProjeto	Numérico	Não	Sim	Da tabela RequisitosUsuario
idModuloProjeto	**Numérico**	Não	Sim	Da tabela ModulosProjeto

Quadro 20 – Campos da tabela RequisitosUsuarioSistema

Tabela Atividades (Quadro 16) – Composição dos requisitos do sistema, detalhando cada atividade necessária para completar o desenvolvimento de determinado requisito do sistema.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idAtividade	Numérico	Sim	Não	
idUsuario	Numérico	Não	Sim	Da tabela Uusarios
idProjeto	Numérico	Não	Sim	Da tabela RequisitosUsuario
idModuloProjeto	Numérico	Não	Sim	Da tabela ModulosProjeto
idRequisitoSistema	Numérico	Não	Sim	Da tabela RequisitosSistema
nome	Texto	Não	Não	
descricao	Texto	Não	Não	
dataInicioPrevista	Data	Não	Não	
dataInicioRealizada	Data	Não	Não	
dataFimPrevista	Data	Não	Não	
dataFimRealizada	Data	Não	Não	
tempoEstimado	Numérico	Não	Não	
tempoUtilizado	Numérico	Não	Não	

Quadro 21 – Campos da tabela Atividades

Tabela Tipo_Classe (Quadro 17) – Tipo da classe, utilizada para categorizar as classes definidas para o sistema.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idTipoClasse	Numérico	Não	Sim	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 22 – Campos da tabela Tipo_Classes

Tabela Classes (Quadro 18) – Classes criadas pelos desenvolvedores.

Campo	Tipo	Chave primária	Chave estrangeira	Valor padrão	Observações
idClasse	Numérico	Sim	Não		
idTipoClasse	Numérico	Não	Sim		Da tabela Tipo_Classes
nome	Texto	Não	Não		
descricao	Texto	Não	Não		
observacao	Texto	Não	Não		
dataCriacao	Data	Não	Não	Data atual	Do sistema operacional

Quadro 23 – Campos da tabela Classes

Tabela Tipo_CasoDeUso (Quadro 19) – Tipo de caso de uso especificado.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idTipoCasoDeUso	Numérico	Sim	Não	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 24 – Campos da tabela Tipo_CasosDeUso

Tabela CasosDeUso (Quadro 20) – Caso de uso definidos para o sistema.

Campo	Tipo	Chave primária	Chave estrangeira	Valor padrão	Observações
idCasoDeUso	Numérico	Sim	Não		
idUsuario	Numérico	Não	Sim		Da tabela Usuarios
idTipoCasoDeUso	Numérico	Não	Sim		Da tabela Tipo_CasoDeUso
nome	Texto	Não	Não		
descricao	Texto	Não	Não		
observacao	Texto	Não	Não		
dataCriacao	Data	Não	Não	Data atual	Do sistema operacional

Quadro 25 – Campos da tabela CasosDeUso

Tabela Tabelas (Quadro 21) – Armazena o cadastro das tabelas definidas para o sistema.

Campo	Tipo	Chave primária	Chave estrangeira	Valor padrão	Observações
idTabela	Numérico	Sim	Não		
idUsuario	Numérico	Não	Sim		Da tabela Usuario

nome	Texto	Não	Não		
descricao	Texto	Não	Não		
dataCriacao	Data	Não	Não	Data atual	Do sistema operacional

Quadro 26 – Campos da tabela Tabelas

Tabela Relatorios_Consultas (Quadro 21) – armazena o cadastro das consultas e relatórios definidos para o sistema.

Campo	Tipo	Chave primária	Chave estrangeira	Valor padrão	Observações
idRelatorio_Consulta	Numérico	Sim	Não		
idUsuario	Numérico	Não	Sim		Da tabela Usuario
Nome	Texto	Não	Não		
Descrição	Texto	Não	Não		
dataCriacao	Data	Não	Não	Data atual	Do sistema operacional

Quadro 27 – Campos da tabela Relatorios_Consultas

Tabela Relatorios_Consultas_Usos (Quadro 21) – para o registro do relacionamento entre consultas e entre relatórios e consultas.

Campo	Tipo	Chave primária	Chave estrangeira	Valor padrão	Observações
idRelatorio_Consulta	Numérico	Sim	Não		Da tabela Relatorios_Consultas
idTabela	Numérico	Não	Sim		Da tabela Tabelas

Quadro 28 – Campos da tabela Relatorios_Consultas_Usos

Tabela ComposicoesCasoDeUso (Quadro 22) – Detalhamento do caso de uso, o qual será relacionado com um requisito do sistema e uma tabela do banco de dados.

Campo	Tipo	Chave primária	Chave estrangeira	Observações
idComposicaoCasosDeUso	Numérico	Sim	Não	
idCasoDeUso	Numérico	Não	Sim	Da tabela CasosDeUso
idRequisitoSistema	Numérico	Não	Sim	Da tabela RequisitosSistema
idClasse	Numérico	Não	Sim	Da tabela Classes
idTabela	Numérico	Não	Sim	Da tabela Tabelas
idRelatorio_Consulta	Numérico	Não	Sim	Da tabela Relatorios_Consultas
idUsuario	Numérico	Não	Sim	Da tabela Usuarios
dataVinculo	Data	Não	Não	Do sistema operacional

observacao	Texto	Não	Não	

Quadro 29 – Campos da tabela ComposicaoCasosDeUso

Tabela Tipos_Artefatos (Quadro 24) – contém os dados de cadastro de tipos de artefatos.

Campo	**Tipo**	**Chave primária**	**Chave estrangeira**	**Observações**
idTipoArtefato	Numérico	Sim	Não	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	

Quadro 30 – Campos da tabela Tipo_Artefatos

Tabela Tipos_Artefatos (Quadro 25) – contém os dados de cadastro de tipos de artefatos.

Campo	**Tipo**	**Chave primária**	**Chave estrangeira**	**Observações**
idArtefato	Numérico	Sim	Não	
nome	Texto	Não	Não	
descricao	Texto	Não	Não	
artefato	Arquivo	Não	Não	Arquivo anexado
idTipoArtefato	Numérico	Não	Não	Da tabela TipoArtefato
idUsuario	Numérico	Não	Não	Da tabela Usuario

Quadro 31 – Campos da tabela Tipo_Artefatos

5.3 APRESENTAÇÃO DO SISTEMA

Ao acessar o aplicativo, tanto o professor quanto o administrador e o aluno tem como tela inicial o *login* apresentado na Figura 5.

Figura 5 – Tela de *login* no sistema

A Figura 6 apresenta a tela inicial do cadastro e de visualização de projetos. Essa tela é uma tabela (*grid*) com linhas selecionáveis, que ao serem clicadas retornam à página do projeto, apresentando seus módulos cadastrados. Os botões à direita permitem excluir ou editar o respectivo cadastro.

Figura 6 – Tela de lista de projetos

Ao adicionar uma equipe, o professor já pode inserir os alunos em seus respectivos papéis, que ao serem adicionados já são apresentados na tabela de membros. O professor pode também adicionar apenas a equipe e depois ao selecionar o projeto e clicar no botão alterar para adicionar seus membros. Nesse caso, o professor pode adicionar os membros e clicar em salvar para armazenar os membros adicionados ou cancelar e as alterações não serão salvas. A Figura 7 apresenta a tela compor uma equipe.

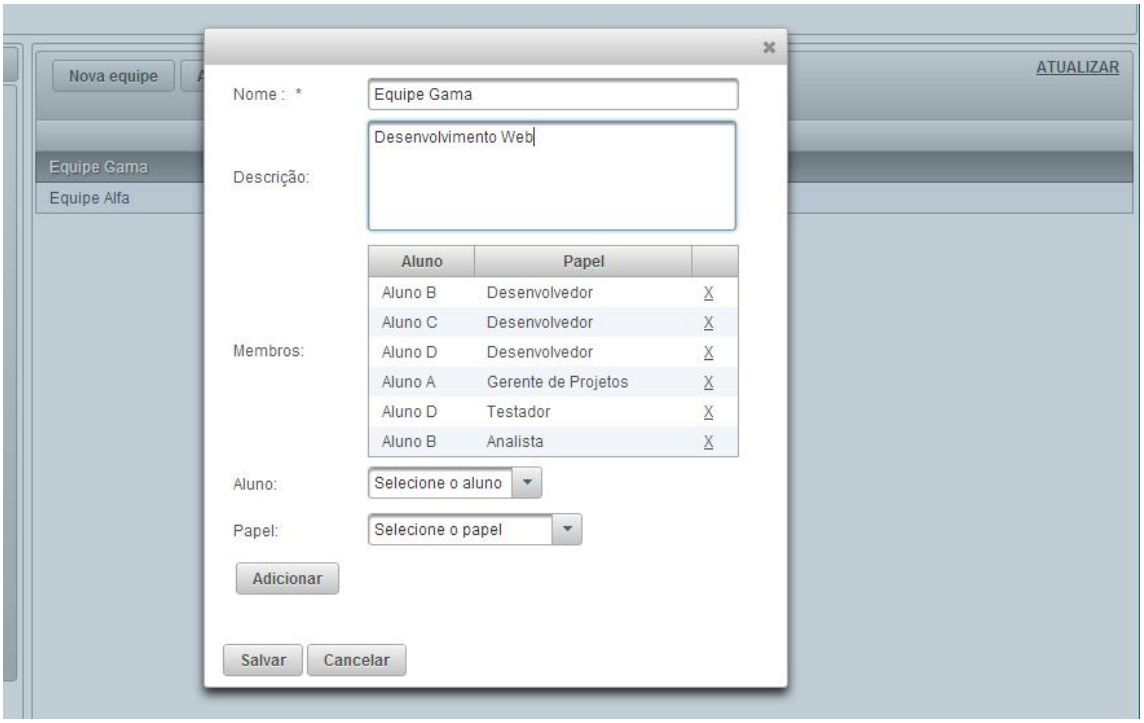

Figura 7 – Compor equipe

Quando o professor clicar em novo projeto, ele já é adicionado como participante no respectivo projeto. Nessa tela também há a opção de adicionar mais professores. Isso também pode ser feito ao selecionar o projeto e clicar em alterar. A tela de cadastro de um novo projeto é apresentada na Figura 8.

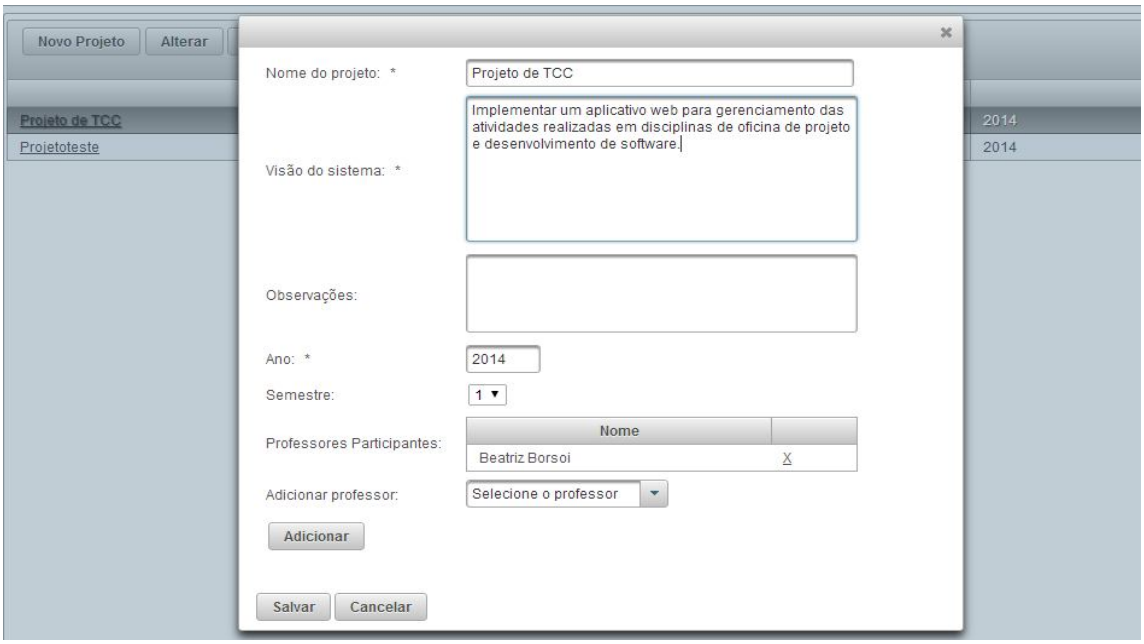

Figura 8 – Tela de cadastro de um novo projeto

Na Figura 9 está um exemplo de visualização quando um projeto já está cadastrado. Por ser a visualização dos módulos há um *link* “Atividades” que redireciona o usuário para a tela de listagens das atividades e os requisitos do usuário que compõem o módulo selecionado.

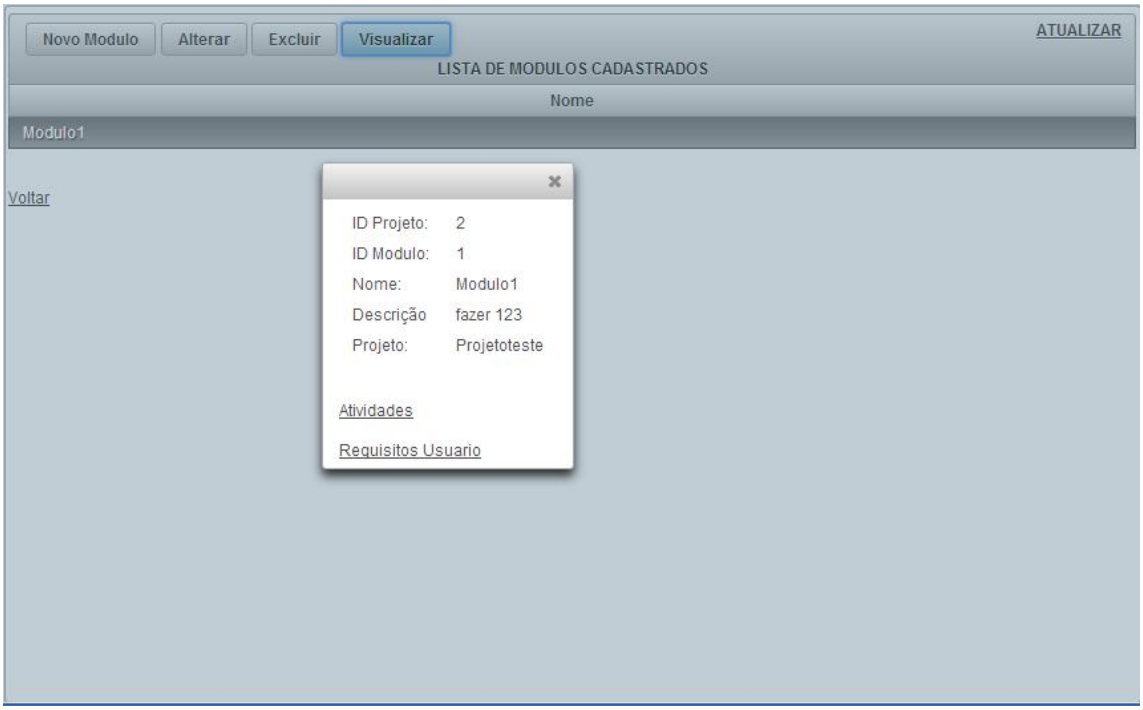

Figura 9 – Tela de visualização de projeto cadastrado

Na Figura 10 é apresentada uma listagem simples de usuários. Nessa listagem quando nenhum item está selecionado, os botões para edição, visualização e exclusão ficam desabilitados. Quando um usuário é selecionado esses botões são habilitados possibilitando realizar as respectivas operações com o registro selecionado.

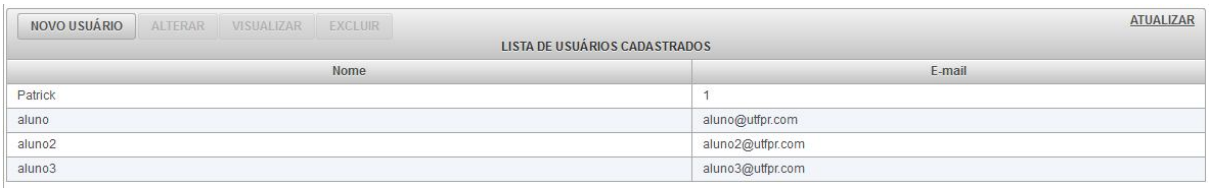

Figura 10 – Tela de listagem de usuários cadastrados

A Figura 11 é do cadastro de artefatos que compõem o sistema. Um artefato é um diagrama de entidades e relacionamentos, um diagrama de classes, um *script* de banco de dados, dentre outros. Quando o aluno cadastra um artefato o sistema informa tipos de artefatos que foram cadastrados pelo professor. O tipo é utilizado para categorizar o artefato.

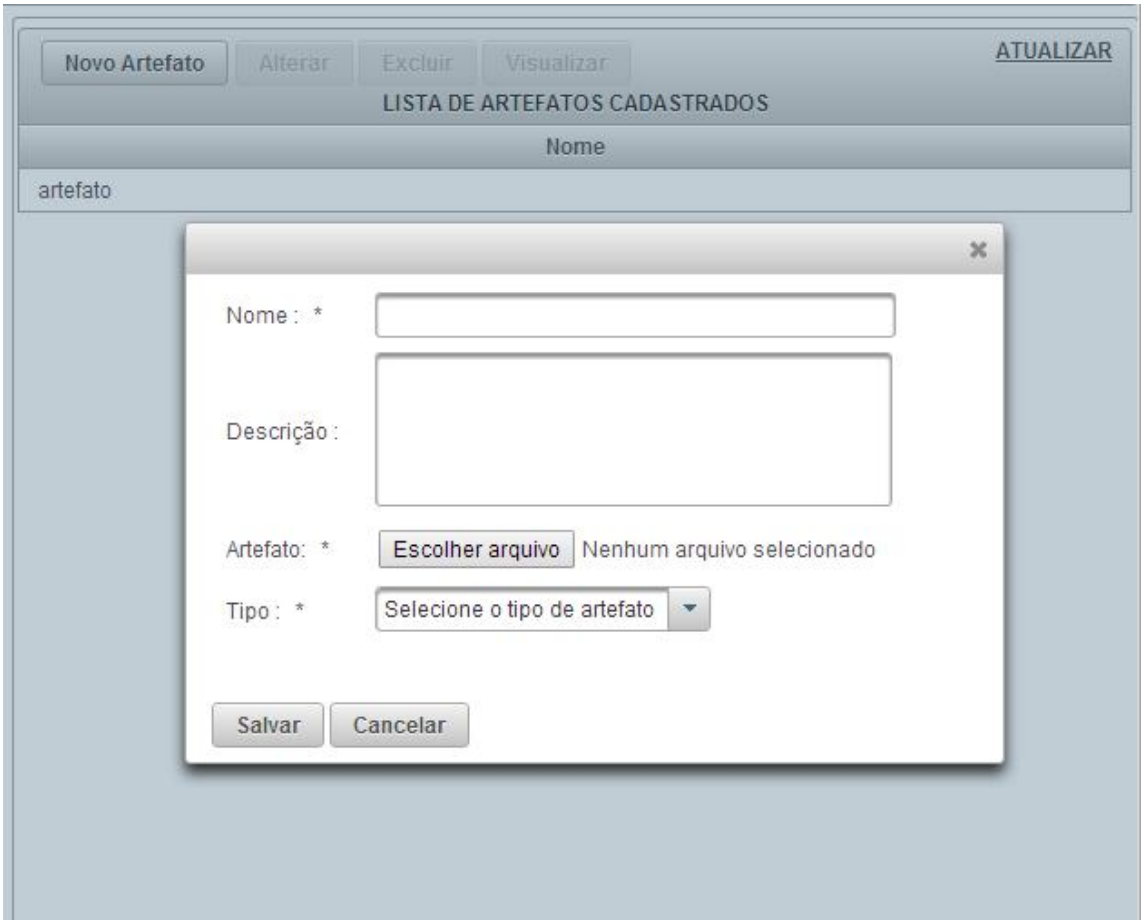

Figura 11 – Tela de cadastro de um novo artefato

Na Figura 12 está a tela para o acesso aos cadastros de vínculos entre o caso de uso e as tabelas, caso de uso e classe, caso de uso e requisitos do sistema, caso de uso e artefatos, e caso de uso e relatórios/consultas.

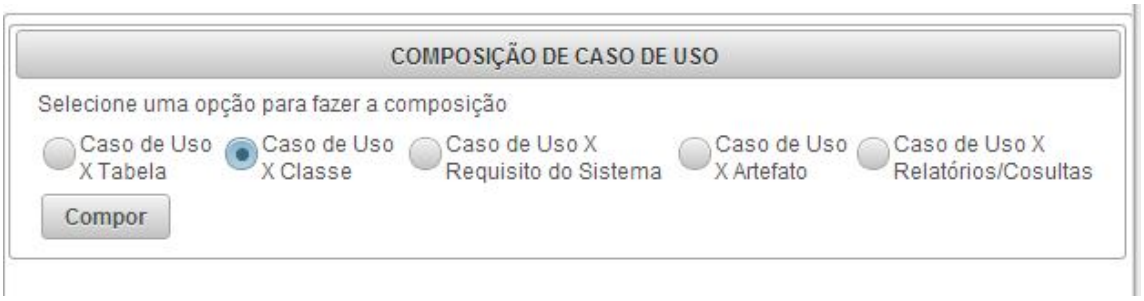

Figura 12 – Tela para acessar os cadastros de vínculos entre artefatos

Na Figura 13 está a tela para a composição de caso de uso e tabelas, sendo que esta mesma interface é apresentada nos demais casos.

Figura 13 – Tela para vínculo entre casos de uso de tabelas

5.4 IMPLEMENTAÇÃO DO SISTEMA

O projeto foi inicialmente criado por um projeto Java Web, Aplicação Web, mas com o decorrer dos estudos foi verificado que era mais produtivo usar Aplicação Web pelo Maven, pois esse gerencia os pacotes de bibliotecas de forma mais efetiva. Isso porque os pacotes necessários são baixados automaticamente da Internet e são implantados na aplicação, evitando, assim, problemas por falta de pacotes dependentes. A troca de versão dos pacotes também é facilitada, sendo necessário apenas acessar o arquivo *Pom.xml* e escolher a versão desejada da dependência. A Figura 11 apresenta a organização do projeto.

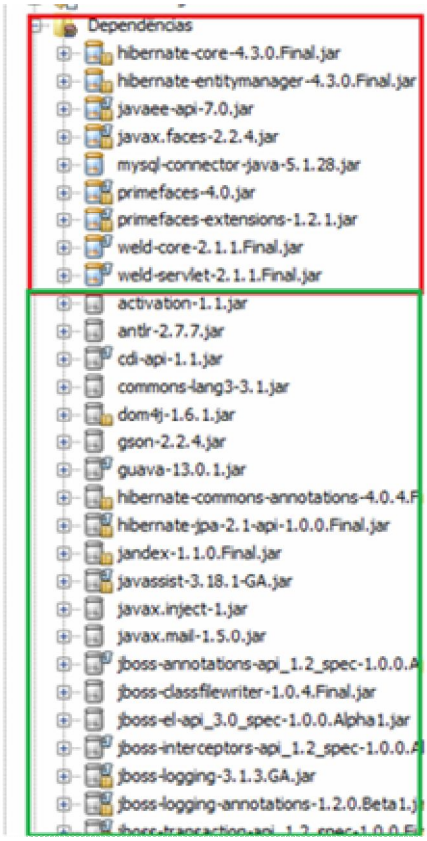

Figura 14 – Organização do projeto

Na Figura 11, os pacotes destacados em vermelho são dependências diretas do projeto, e os pacotes marcados em cor verde indicam dependências indiretas, ou seja, são dependências das dependências.

O arquivo pom.xml, na Listagem 1, armazena as configurações de dependências do projeto, assim como informações sobre o arquivo compilado e configurações de compilação ficou da seguinte forma.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>br.edu.utfpr</groupId>
  <artifactId>ProjetoTCC</artifactId>
  <version>1.0-Alfa</version>
  <packaging>war</packaging>

  <name>ProjetoTCC</name>

  <properties>
    <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.primefaces.extensions</groupId>
```

```
        <artifactId>primefaces-extensions</artifactId>
        <version>1.2.1</version>
    </dependency>
    <dependency>
        <groupId>org.jboss.weld</groupId>
        <artifactId>weld-core</artifactId>
        <version>2.1.1.Final</version>
    </dependency>
    <dependency>
        <groupId>org.jboss.weld.servlet</groupId>
        <artifactId>weld-servlet</artifactId>
        <version>2.1.1.Final</version>
    </dependency>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>4.0</version>
        <type>jar</type>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.28</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish</groupId>
        <artifactId>javax.faces</artifactId>
        <version>2.2.4</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.3.0.Final</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.3.0.Final</version>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>7.0</version>
        <type>jar</type>
        <scope>provided</scope>
    </dependency>
</dependencies>

<build>
    <sourceDirectory>
        ${project.basedir}/src
    </sourceDirectory>
    <testSourceDirectory>
        ${project.basedir}/test
    </testSourceDirectory>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <source>1.7</source>
```

```
            <target>1.7</target>
            <compilerArguments>
              <endorseddirs>${endorsed.dir}</endorseddirs>
            </compilerArguments>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-war-plugin</artifactId>
          <version>2.4</version>
          <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-dependency-plugin</artifactId>
          <version>2.8</version>
          <executions>
            <execution>
              <phase>validate</phase>
              <goals>
                <goal>copy</goal>
              </goals>
              <configuration>
                <outputDirectory>${endorsed.dir}</outputDirectory>
                <silent>true</silent>
                <artifactItems>
                  <artifactItem>
                    <groupId>javax</groupId>
                    <artifactId>javaee-web-api</artifactId>
                    <version>7.0</version>
                    <type>jar</type>
                  </artifactItem>
                </artifactItems>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
    <repositories>
      <repository>
        <url>http://repository.primefaces.org/</url>
        <id>PrimeFaces-maven-lib</id>
        <layout>default</layout>
        <name>Repository for library PrimeFaces-maven-lib</name>
      </repository>
    </repositories>
</project>
```

Listagem 1 – Arquivo pom.xml

Os arquivos foram organizados na forma como apresentado na Figura 12. De acordo com essa Figura, os pacotes br.edu.utfpr.projetotcc contém todo o código Java da aplicação, os pacotes main.resources armazenam recursos para geração de *logs* e arquivos de configurações de persistência de dados do Hibernate, enquanto que o pacote main.webapp armazena as páginas e fragmentos das páginas da aplicação, assim como configurações do JSF.

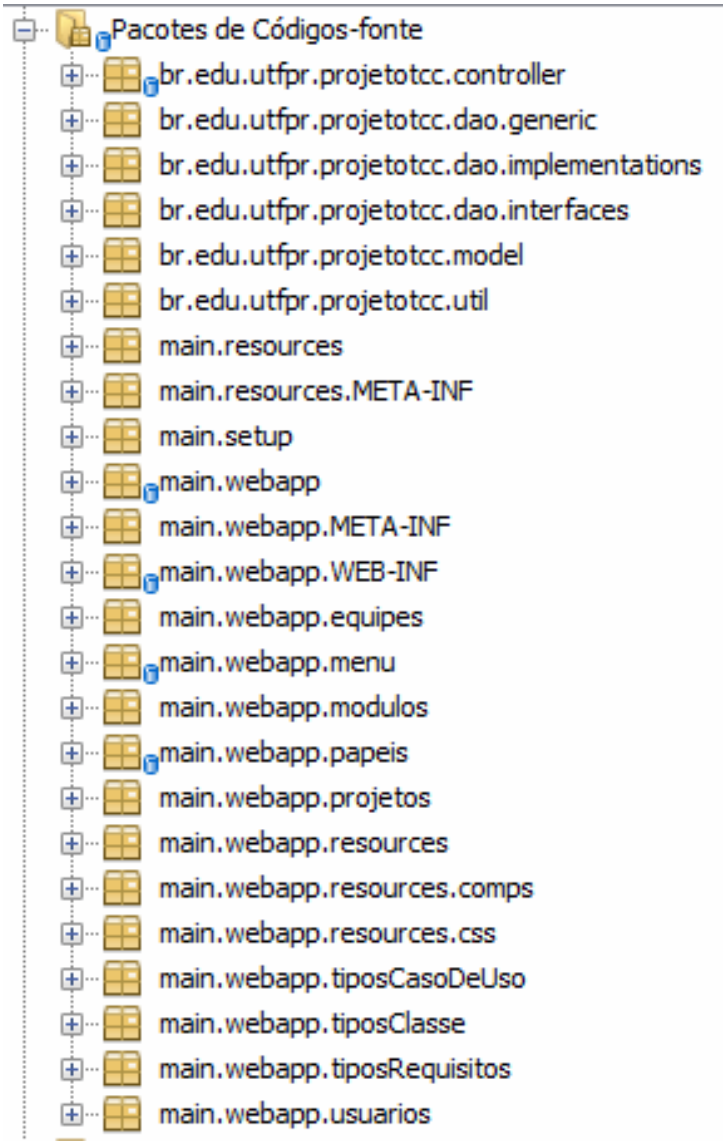

Figura 15 – Organização dos arquivos

A página inicial, localizada no pacote main.webapp da aplicação, ficou codificada da forma como apresentado na Listagem 2.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

  <h:head>
    <h:outputStylesheet library="css" name="pfcrud.css"/>
    <title>Gerenciamento de Projetos</title>
  </h:head>
  <h:body>
```

```
    <!-- p:layout é responsável pela divisão da página em cabeçalho,
            manu lateral e conteúdo central, cada um destes são os p:layoutUnit-->
    <p:layout fullPage="true" id="page">
      <p:layoutUnit position="north" size="100">
        <h:form id="topMenu" >
          <!-- Inclui um fragmento de página dentro desta, formando somente 1 HTML
            somente-->
          <ui:include src="${mainMenuBean.topMenu}"/>
        </h:form>
      </p:layoutUnit>

      <p:layoutUnit position="west" size="230" header="Menu">
        <h:form id="leftMenu">
          <ui:include src="${mainMenuBean.leftMenu}"/>
        </h:form>
      </p:layoutUnit>

      <p:layoutUnit position="center">
        <h:form id="centerContent">
          <ui:include src="${mainMenuBean.centerContentPage}"/>
        </h:form>

      </p:layoutUnit>
    </p:layout>

    <!-- Caixa de diálogo usado em todas as confirmações da aplicação-->
    <p:confirmDialog global="true" width="350" showEffect="fade" hideEffect="explode">
      <h:form>
        <p:commandButton value="Sim" type="button" styleClass="ui-confirmdialog-yes" icon="ui-icon-
check"/>
        <p:commandButton value="Não" type="button" styleClass="ui-confirmdialog-no" icon="ui-icon-close"/>
      </h:form>
    </p:confirmDialog>
    <!-- Listener que é chamado a cada execução AJAX da aplicação, executando uma ação
      quando inicia o processamento, quando encerra com sucesso ou com erro-->
    <p:ajaxStatus onstart="PF('statusDialog').show();"
            onsuccess="PF('statusDialog').hide();"
            onerror="PF('statusDialog').hide();
            PF('failDialog').show();"/>

    <!-- Caixa de diálogo chamado pelo p:ajaxStatus para mostrar uma imagem animada
      enquanto o processamento AJAX é executado pelo servidor-->
    <p:dialog modal="true" widgetVar="statusDialog" header="Aguarde..." showEffect="puff"
        draggable="false" closable="false" resizable="false">
      <ui:include src="/resources/preloader_JS.html"/>
    </p:dialog>

    <!-- Caixa de diálogo chamado pelo p:ajaxStatus para avisar que o processamento
      AJAX falou na requisição ou processamento-->
    <p:dialog modal="true" widgetVar="failDialog" header="Ocorreu um problema..."
        closable="true" resizable="false">
      Desculpe, ocorreu um problema na execução da requisição
    </p:dialog>

  </h:body>
</html>
```

Listagem 2 – Página inicial do pacote main.webapp

O código para a implementação do menu lateral, incluso no index.xhtml da página é apresentado na Listagem 3.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:p="http://primefaces.org/ui"
   xmlns:pe="http://primefaces.org/ui/extensions"
   xmlns:f="http://java.sun.com/jsf/core">

   <p:menu>
      <!--  O action é o método do lado Servidor que será executado quando
         for dado clique no menu.

            A Propriedade "rendered" ditará a visibilidade do menu, sendo
         que neste caso só será visível quando o usuário estiver logado
         na aplicação

            Após a execução do método do lado Servidor, o conteúdo central
         do formulário será atualizada, trocando então a página que o
         usuário deverá visualizar
      -->
      <p:menuitem value="Atividade" update=":centerContent"
            action="#{mainMenuBean.goToPage('atividades.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Equipe" update=":centerContent"
            action="#{mainMenuBean.goToPage('/equipes/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Papel" update=":centerContent"
            action="#{mainMenuBean.goToPage('/papeis/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Projetos" update=":centerContent"
            action="#{mainMenuBean.goToPage('/projetos/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Usuario" update=":centerContent"
            action="#{mainMenuBean.goToPage('/usuarios/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Tipo de Caso de Uso" update=":centerContent"
            action="#{mainMenuBean.goToPage('/tiposCasoDeUso/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Tipo de Requisito" update=":centerContent"
            action="#{mainMenuBean.goToPage('/tiposRequisitos/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>
      <p:menuitem value="Tipo de Classe" update=":centerContent"
            action="#{mainMenuBean.goToPage('/tiposClasse/index.xhtml')}"
            rendered="#{mainMenuBean.loggedIn}"/>

   </p:menu>

</ui:composition>
```

Listagem 3 – Menu lateral index.xhtml

O código do arquivo header.xhtml incluso também na página inicial é apresentado na Listagem 4.

```
<?xml version='1.0' encoding='UTF-8' ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<ui:composition rendered="#{mainMenuBean.loggedIn}"
            xmlns:ui="http://java.sun.com/jsf/facelets"
            xmlns="http://www.w3.org/1999/xhtml"
            xmlns:h="http://java.sun.com/jsf/html"
            xmlns:p="http://primefaces.org/ui"
            xmlns:pe="http://primefaces.org/ui/extensions"
            xmlns:f="http://java.sun.com/jsf/core">

   <h:outputLabel value="Usuário: #{mainMenuBean.usuarioLogado.nome}"
rendered="#{mainMenuBean.loggedIn}"/>

   <p:commandLink id="logoutButton" style="margin-left: 20px" value="Logout" update=":leftMenu, :topMenu,
:centerContent"
              action="#{mainMenuBean.logout()}" rendered="#{mainMenuBean.loggedIn}"/>

   <br/>
   #{mainMenuBean.msg}

</ui:composition>
```

Listagem 4 – Arquivo header.xhtml

O usuário após clicar no menuItem para acessar o cadastro de papéis chama o método mainMenuBean.goToPage passando por parâmetro a página que deseja acessar, que no caso é o index.xhtml da pasta papéis. A codificação para troca de página é apresentada na Listagem 5.

```
package br.edu.utfpr.projetotcc.controller;

import br.edu.utfpr.projetotcc.dao.interfaces.EquipeDao;
import br.edu.utfpr.projetotcc.model.EquipeUsuario;
import br.edu.utfpr.projetotcc.model.ModuloProjeto;
import br.edu.utfpr.projetotcc.model.Projeto;
import br.edu.utfpr.projetotcc.model.Usuario;
import java.io.IOException;
import java.io.Serializable;
import java.util.List;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import javax.inject.Inject;
import main.resources.Consts;
import static main.resources.Consts.PERMISSAO_ADMINISTRADOR;

//@Named significa que ele será um Bean Gerenciado que poderá ser
//acessado pelas páginas xhtml da aplicação
@Named
//A Anotação @SessionScoped dita o ciclo de vida deste Bean Gerneciado
//o qual esta define que será ativa enquanto a sessão do usuário estiver
//ativa (definida o tempo limite no arquivo web.xml como 30 minutos)
@SessionScoped
public class MainMenuBean implements Serializable {

   private static final long serialVersionUID = 1L;

   //Esta propriedade deverá ficar preenchida somente se o usuário
   //fizer login com sucesso
   private Usuario usuarioLogado;
   private String centerContentPage;
```

```
private String msg = " ";

/*
   A Anotação @Inject define que ao ser instanciado o Bean, será
   instanciado a interface equipeDao pela sua implementação Padrão
*/
@Inject
EquipeDao equipeDao;

public MainMenuBean() {
   this.centerContentPage = "/menu/welcomePage.xhtml";
}

public void goToPage(String page) {
   this.setCenterContentPage(page);
}

public boolean isLoggedIn() {
   return usuarioLogado != null;
}

public String getLeftMenu() {
   return "/menu/left.xhtml";
}

public String getTopMenu() {
   return "/menu/header.xhtml";
}

public String getCenterContentPage() {
   //Ao tentar acessar quaisquer página, verifica se está logado,
   //caso não estiver, é redirecionado para a página de login
   //caso estiver, acessa a páqina requirida, caso não ter definido
   //ainda a página que quiser acessar, é direcionado para a página
   //de boas vindas

   if (usuarioLogado == null) {
      return "login.xhtml";
   } else {
      if (centerContentPage.equals("")) {
         return "/menu/welcomePage.xhtml";
      } else {
         return centerContentPage;
      }

   }
}

public void setCenterContentPage(String centerContentPage) {
   this.centerContentPage = centerContentPage;
}

public Usuario getUsuarioLogado() {
   return usuarioLogado;
}

public void setUsuarioLogado(Usuario usuarioLogado) {
```

```
        this.usuarioLogado = usuarioLogado;
    }

    public void logout() {
        usuarioLogado = null;
    }

    /**
     *
     * Esta função tem como objetivo detalhar a participação do usuário em
     * diferentes projetos, seja ele Aluno ou Professor
     */
    public String getMsg() {

        List<Projeto> lista;

        if (usuarioLogado == null) {
            msg = " ";
        } else if (usuarioLogado.getPermissoes() == PERMISSAO_ADMINISTRADOR) {
            msg = "Administrador";
        } else if (usuarioLogado.getPermissoes() == Consts.PERMISSAO_PROFESSOR) {
            lista = usuarioLogado.getProjetoList();
            if (lista.isEmpty()) {
                msg = "Professor - Sem projetos definidos";
            } else {
                lista = usuarioLogado.getProjetoList();
                msg = "Professor - ";
                for (Projeto p : lista) {
                    msg = msg + p.getNome();
                }
            }
        } else { //Caso for Aluno
            List<EquipeUsuario> listaParticipacoes = usuarioLogado.getEquipeUsuarioList();

            for (EquipeUsuario eu : listaParticipacoes) {
                msg = eu.getPapel().getNome() + " na equipe " + eu.getEquipe().getNome() + " no projeto " ;

                List<ModuloProjeto> listaModulos = eu.getEquipe().getModuloProjetoList();
                for (ModuloProjeto mp : listaModulos) {
                    msg = msg + mp.getProjeto().getNome();
                }
            }
        }
        return msg;
    }

}
```

Listagem 5 – Método mainMenuBean.goToPage

Todos os objetos responsáveis pelo acesso ao banco de dados são os DAOs que estendem do GenericDAO. O código do GenericDAO é apresentado na Listagem 6.

```
package br.edu.utfpr.projetotcc.dao.generic;

import java.util.List;
```

```
public interface GenericDAO<T>{

   void beginTransaction();

   void commitTransaction();

   void save(T object);

   void delete(T object);

   List<T> listAll();

   T findById(Object id);

}
```

Listagem 6 – GenericDAO

E a implementação do GenericDao é apresentada na Listagem 7.

```
package br.edu.utfpr.projetotcc.dao.generic;

import java.io.Serializable;
import java.util.List;
import javax.annotation.Resource;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.PersistenceContextType;
import javax.persistence.PersistenceProperty;
import javax.transaction.HeuristicMixedException;
import javax.transaction.HeuristicRollbackException;
import javax.transaction.NotSupportedException;
import javax.transaction.RollbackException;
import javax.transaction.SystemException;
import javax.transaction.UserTransaction;
import main.resources.ApplicationLogger;

public abstract class GenericDAOImpl<T extends Object> implements GenericDAO<T>, Serializable {

   private static final long serialVersionUID = 1L;

   /*
      @PersistenceContext instancia e define através do arquivo de
      configurações persistence.xml a conexão com o banco de dados,
      define também o ciclo de vida da conexão, que por padrão é por
      transação feita.
   */
   @PersistenceContext
   protected EntityManager entityManager;

   //Instancia o UserTransaction do DAO
   @Resource
   private UserTransaction transaction;

   private final Class<T> persistentClass;

   public GenericDAOImpl(Class<T> persistentClass) {
```

```
    super();
    this.persistentClass = persistentClass;
}

@Override
public void beginTransaction() {
    try {
        transaction.begin();
    } catch (NotSupportedException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    } catch (SystemException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    }
}

@Override

public void commitTransaction() {
    try {
        transaction.commit();
    } catch (RollbackException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    } catch (HeuristicMixedException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    } catch (HeuristicRollbackException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    } catch (SecurityException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    } catch (IllegalStateException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    } catch (SystemException ex) {
        ApplicationLogger.severe(GenericDAOImpl.class, ex);
    }
}

@Override
@SuppressWarnings("unchecked")
public T findById(Object id) {
    beginTransaction();
    Object row = entityManager.find(persistentClass, id);
    commitTransaction();
    return (T) row;
}

@Override
public void save(T object) {
    beginTransaction();
    entityManager.merge(object);
    commitTransaction();
}

@Override
public void delete(T object) {
    beginTransaction();
    entityManager.remove(entityManager.merge(object));
    commitTransaction();
}
```

```
    @Override
    public List<T> listAll() {
        List<T> lista = entityManager.createNamedQuery(
                persistentClass.getSimpleName()+".findAll",persistentClass).getResultList();
        return lista;
    }

}
```

Listagem 7 – Implementação do GenericDAO

A Listagem 8 apresenta a instanciação para PapelDao.

```
package br.edu.utfpr.projetotcc.dao.interfaces;


import br.edu.utfpr.projetotcc.dao.generic.GenericDAO;
import br.edu.utfpr.projetotcc.model.Papel;


public interface PapelDao extends GenericDAO<Papel>{
}
```

Listagem 8 – Instanciação para PapelDao

A implementação para PapelDao está na Listagem 9.

```
package br.edu.utfpr.projetotcc.dao.implementations;

import br.edu.utfpr.projetotcc.dao.interfaces.PapelDao;
import br.edu.utfpr.projetotcc.dao.generic.GenericDAOImpl;
import br.edu.utfpr.projetotcc.model.Papel;


public class PapelDaoImpl extends GenericDAOImpl<Papel> implements PapelDao{
    private static final long serialVersionUID = 1L;

    public PapelDaoImpl() {
        super(Papel.class);
    }
}
```

Listagem 9 – Implementação do PapelDao

A intermediação da comunicação do usuário com o Dao é realizada por meio da classe Bean, cuja codificação é apresentada na Listagem 10.

```
package br.edu.utfpr.projetotcc.controller;

import br.edu.utfpr.projetotcc.dao.generic.GenericBean;
import br.edu.utfpr.projetotcc.dao.interfaces.PapelDao;
import br.edu.utfpr.projetotcc.model.Papel;
import javax.faces.view.ViewScoped;
import javax.inject.Inject;
```

```
import javax.inject.Named;


@Named
@ViewScoped
public class PapelBean extends GenericBean<Papel>{

   private static final long serialVersionUID = 4058842017328831331L;

   public PapelBean() {
   }

   /*
      Instancia o dao na classe ancestral
      o @Inject neste caso vai instanciar o PapelDao pela sua classe
      de implementação padrão: PapelDaoImpl
   */
   @Inject
   public PapelBean(PapelDao dao) {
      super(dao);
   }

   @Override
   public void addItem() {
      super.addItem(Papel.class);
   }
}
```

Listagem 10 – Classe bean para comunicação do usuário com Dao

A classe PapelBean estende uma classe Genérica que contém as funções de CRUD. O código dessa classe é apresentado na Listagem 11.

```
package br.edu.utfpr.projetotcc.dao.generic;

import java.io.Serializable;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.faces.context.FacesContext;
import main.resources.ApplicationLogger;

public abstract class GenericBean<T extends Object> implements Serializable{
   private static final long serialVersionUID = 1L;

   /*
      Declaração genérica do dao pra ser instanciado por Classes herdeiras
      não é possível instanciá-lo através do @Inject pois existem várias
      classes que implementam essa interface, por isso no construtor da
      classe herdeira inicializa-se esta propriedade com o Tipo correto
   */
   protected GenericDAO<T> dao;
   private T selectedItem;

   public GenericBean() {
```

```
}

public GenericBean(GenericDAO<T> dao) {
    this.dao = dao;
}

public T getSelectedItem() {
    return selectedItem;
}

public void setSelectedItem(T selectedItem) {
    this.selectedItem = selectedItem;
}
public void save(){
    dao.save(selectedItem);
    selectedItem = null;
}

public List<T> getListItems() {
    return dao.listAll();
}


public void delete() {
    dao.delete(selectedItem);
    selectedItem = null;
}


public abstract void addItem();

@SuppressWarnings("unchecked")
protected void addItem(Class<T> itemClass) {
    try {
        this.selectedItem = itemClass.newInstance();
    } catch (InstantiationException | IllegalAccessException ex) {
        Logger.getLogger(GenericBean.class.getName()).log(Level.SEVERE, null, ex);
    }

}

/*
    A anotação @PostConstruct permite que um trecho de código seja
    executado quando o ciclo de vida do Bean gerenciado é iniciado
*/
@PostConstruct
private void init(){
    ApplicationLogger.information(this.getClass(), "Inicializado instância da Classe: " +
            this.getClass().getSimpleName() +
            " na sessão: " +
            FacesContext.getCurrentInstance().getExternalContext().getSessionId(false));
}

/*
    A anotação @PreDestroy permite que um trecho de código seja
    executado quando o ciclo de vida do Bean gerenciado é encerrado
*/
```

```
    @PreDestroy
    private void destroy(){
        ApplicationLogger.information(this.getClass(), "Destruído instância da Classe: " +
                this.getClass().getSimpleName() +
                " na sessão: " +
                FacesContext.getCurrentInstance().getExternalContext().getSessionId(false));
    }

}
```

Listagem 11 – Classe PapelBean

O JSF interpreta a página apresentada na Listagem 12 para mostrar ao usuário a lista de Papéis (/papeis/papeis.xhtml).

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">

    <ui:include src="/papeis/list.xhtml"/>
    <ui:include src="/papeis/view.xhtml"/>
    <ui:include src="/papeis/edit.xhtml"/>
</ui:composition>
```

Listagem 12 – Página papeis.xhtml

A página papeis.xhtml resulta da reunião de outros três fragmentos de páginas apresentados nas Listagens 13, 14 e 15.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
            xmlns:h="http://java.sun.com/jsf/html"
            xmlns:f="http://java.sun.com/jsf/core"
            xmlns:p="http://primefaces.org/ui"
            xmlns:ui="http://java.sun.com/jsf/facelets"
            xmlns:c="http://java.sun.com/jsf/composite/comps">


    <h:form id="frmprincipal" prependId="false" >

        <p:dataTable  id="tabela"
                    var="papel"
                    value="#{papelBean.listItems}"
                    selection="#{papelBean.selectedItem}"
                    selectionMode="single"
                    rowKey="#{papel.idPapel}">

            <f:facet name="header">
                <div class="header-crud">
                    <p:commandButton id="novoRegistro"
                                value="Novo Papel"
                                actionListener="#{papelBean.addItem()}"
                                update=":centerContent:form:formDetalhes"
                                oncomplete="PF('dlgformDetalhes').show()"/>
                    <p:commandButton id="cmdAlterar"
                                value="Alterar"
                                update=":centerContent:form:formDetalhes"
                                oncomplete="PF('dlgformDetalhes').show()"
```

```
                    disabled="#{empty papelBean.selectedItem}"/>
            <p:commandButton id="view"
                    value="Visualizar"
                    update=":centerContent:PapelView"
                    oncomplete="PF('dlgpapelViewDlg').show()"
                    disabled="#{empty papelBean.selectedItem}"/>
            <p:commandButton id="cmdExcluir"
                    value="Excluir"
                    actionListener="#{papelBean.delete}"
                    update="tabela"
                    disabled="#{empty papelBean.selectedItem}">
              <p:confirm header="Confirmação" message="Confirma a exclusão?" />
            </p:commandButton>
            <p:commandLink id="atualizar"
                    value="Atualizar"
                    update="tabela"
                    style="float: right"/>
          </div>
        </f:facet>

        <p:ajax event="rowSelect"   update="novoRegistro cmdAlterar view cmdExcluir" />
        <p:ajax event="rowUnselect" update="novoRegistro cmdAlterar view cmdExcluir"/>

        <f:facet name="header">Lista de Papéis</f:facet>
        <p:column headerText="Nome" width="40%">
          <h:outputText value="#{papel.nome}"  />
        </p:column>
        <p:column headerText="Visão do Sistema">
          <h:outputText value="#{papel.descricao}"/>
        </p:column>
      </p:dataTable>
    </h:form>


</ui:composition>
```

Listagem 13 – Fragmento de página List.xhtml

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
            xmlns:h="http://java.sun.com/jsf/html"
            xmlns:f="http://java.sun.com/jsf/core"
            xmlns:p="http://primefaces.org/ui"
            xmlns:ui="http://java.sun.com/jsf/facelets"
            xmlns:c="http://java.sun.com/jsf/composite/comps">

    <!-- c:formulario é um componente composto criado para esta aplicação -->
    <c:formulario id="form"
              formName="formDetalhes"
              salvar="#{papelBean.save()}"
              tabelaAtualizar=":centerContent:frmprincipal:tabela"
              width="500">
      <p:panelGrid id="pnlDetalhes" columns="2" >
        <h:outputText value="Papel: "/>
        <p:inputText id="nome" value="#{papelBean.selectedItem.nome}"/>
        <h:outputText value="Decrição:"/>
        <p:editor widgetVar="desc" value="#{papelBean.selectedItem.descricao}"
```

```
                controls="bold italic underline strikethrough subscript superscript | font size style | color highlight removeformat | bullets numbering | outdent indent | alignleft center alignright justify"/>
        </p:panelGrid>
    </c:formulario>
</ui:composition>
```

Listagem 14 – Fragmento de página Edit.xhtml

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:c="http://java.sun.com/jsf/composite/comps">

    <p:dialog
            widgetVar="dlgpapelViewDlg"
            modal="true"
            resizable="false"
            appendToBody="true"
            header="#{papelBean.selectedItem.nome}"
            closeOnEscape="true">

        <h:form id="PapelView">

            <h:panelGroup id="display">
                <p:panelGrid columns="2" rendered="#{papelBean.selectedItem != null}">
                    <h:outputText value="ID: "/>
                    <h:outputText value="#{papelBean.selectedItem.idPapel}"/>
                    <h:outputText value="Nome: "/>
                    <h:outputText value="#{papelBean.selectedItem.nome}"/>
                     <h:outputText value="Descrição: "/>
                     <h:outputText value="#{papelBean.selectedItem.descricao}"/>
                </p:panelGrid>
            </h:panelGroup>
        </h:form>
    </p:dialog>
</ui:composition>
```

Listagem 15 – Fragmento de página View.xhtml

Na página Edit.xhtml (Listagem 16) está a declaração de um componente composto que é personalizado para edição do registro.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:cc="http://java.sun.com/jsf/composite"
    xmlns:p="http://primefaces.org/ui"
    xmlns:h="http://java.sun.com/jsf/html">

    <!--
        Declaração das propriedades do componente
    -->
    <cc:interface>
```

```
    <cc:attribute name="formName" required="true"/>
    <cc:attribute name="width" default="500"/>
    <!-- method-signature é basicamente uma restrição de quais valores a
        propriedade pode receber
    -->
    <cc:attribute name="salvar" method-signature="void action()" required="true" />
    <cc:attribute name="tabelaAtualizar" />
  </cc:interface>

  <cc:implementation>
    <p:dialog id="dialog" header="" widgetVar="dlg#{cc.attrs.formName}"
          resizable="false"        modal="true"        showEffect="fade"        width="#{cc.attrs.width}"
appendTo="@(body)">

      <h:form prependId="false" id="#{cc.attrs.formName}">

        <cc:insertChildren/>
        <br/><br/>
        <p:commandButton value="Salvar" id="salvar"
                action="#{cc.attrs.salvar}"
                oncomplete="PF('dlg#{cc.attrs.formName}').hide()"
                update="#{cc.attrs.tabelaAtualizar}"
                process="@form">
        </p:commandButton>
        <p:commandButton value="Cancelar" id="cancelar"
                oncomplete="PF('dlg#{cc.attrs.formName}').hide()"
                update="#{cc.attrs.tabelaAtualizar}">
          <p:confirm header="Cancelamento" message="Deseja descartar as alterações?" />
        </p:commandButton>


      </h:form>
    </p:dialog>

  </cc:implementation>


</html>
```

Listagem 16 – Página Edit.xhtml

5.3 APRESENTAÇÃO DO SISTEMA

Na implementação dos cadastros verificou-se a necessidade de generalizar as classes de *Data Access Objects* (DAO) e as páginas *web*, com o uso de componentes que pudessem aumentar a produtividade e o reuso das classes. O projeto estava sendo implementado usando as anotações: @ManagedBean, que permitem que as páginas JSF instanciem e acessem os métodos e as propriedades das classes com esta anotação. E com a anotação @ViewScoped que define o ciclo de vida dos *beans* para estarem instanciados somente durante a visualização da página e depois destruídos. Contudo essas anotações estão se tornando obsoletas e

futuramente serão descontinuadas. Então foi necessário utilizar as anotações @Named e @ViewScoped (do pacote javax.faces.view.ViewScoped). Essas anotações fazem parte da especificação *Java* de injeção de dependências e contextos (CDI) cujo uso é possível por meio do Weld. As classes Beans faziam acesso as classes de *Data Access Objects* (DAO) por meio de um DAOFactory, os quais instanciavam as interfaces para uso, mas a dependência de manutenção desta classe a cada Entidade do Banco de Dados criada fez com que fosse preferido o uso da anotação @Inject provida do *Framework* CDI, os quais substituíam completamente o uso da Classe DAOFactory, por simplesmente uma anotação antes da declaração do uso da Interface DAO em Beans Gerenciados.

A estrutura das páginas foi alterada em relação ao inicialmente projetado e sendo implementado para que cada tela seja apenas um fragmento da página, preenchendo o conteúdo central da página, mantendo os menus laterais e superiores fixos durante o acesso ao sistema. Ainda foi verificado que era possível melhorar o controle sobre os pacotes java usados no projeto, deixando-o facilmente exportável através da ferramenta de gerenciamento de projetos Maven. Essa ferramenta contém um documento *Extensible Markup Language* (XML) que define de quais pacotes java o projeto depende, fazendo *download* automático do conteúdo e suas dependências e implantação no servidor *web*, mas para isto foi necessário migrar todos os códigos-fonte para o padrão de pastas predefinidas.

Após a adequação do projeto no Maven foi efetuada engenharia reversa do banco de dados, que consiste em abstrair o modelo relacional já existente no banco de dados e mapeá-los como objetos dentro da Aplicação por meio do *Java Persistence API* (JPA). Criando, assim, as classes e seus relacionamentos de forma orientada a objetos, enquanto o *Java Persistence API* (JPA) se encarrega de transformá-los a entidades relacionais do banco de dados. Após isso foi verificado *memory-leaks*, que são áreas de memória instanciadas pelo servidor Tomcat e não eram liberadas na finalização da aplicação. Isso era causado pelo Hibernate em decorrência das conexões do banco de dados, que eram abertas e nunca fechadas.

Também foi estudada a especificação *Java Persistence API*, que é uma especificação *Java* para mapeamento objeto/relacional e *Java Transaction API* (JTA), responsável pelo controle de transações entre a aplicação e o banco de dados, eliminando o HibernateUtil e usando injeção do EntityManager, classe responsável pelas operações *CRUD* (*Create, Retrieve, Update, Delete)* e pela conexão em si com o Banco de Dados e UserTransaction, classe responsável pelo controle transacional das operações com o banco de dados. Esta mudança em eliminar o HibernateUtil e inserir o EntityManager e UserTransaction deixou a

aplicação usando os recursos principais que a especificação *Java Persistence API e Java Transaction API* sugere. Para isso foi necessário fazer uso de um servidor de aplicações *web Java Enterprise Edition.*

Glassfish foi escolhido por ser o único que oferecia compatibilidade completa ao modelo *Enterprise Java Beans* (EJB), JTA, JPA, *Controller Dependency Injection* (CDI), JSF e com o Netbeans 7.4. Além do Glassfish foi estudado o TomEE, mas o excesso de configurações necessárias para que todas as tecnologias sendo utilizadas trabalhassem em conjunto e a falta de documentação sobre o uso de cada uma dessas tecnologias comprometeu a sua escolha em relação ao Glassfish. E, mesmo porque, o WildFly ainda não é suportado pelo Netbeans e também não está em versão final.

Um modelo de GenericBeans que é um *plugin* do Netbeans: Primefaces *Create, Retrieve, Update, Delete* (CRUD) *generator* também foi utilizado. Esse modelo implementa funções genéricas de listagem, visualização, criação, edição e exclusão de entidades do banco de dados, facilitando a implementação das telas.

Por meio do Hibernate é possível mapear as entidades do banco de dados para classes na aplicação. A especificação *Java Persistense API (JPA)* auxilia na criação dessas classes por meio de *Wizards* e do *Java Transaction API (JTA)* provida pelo próprio *Glassfish* que possibilita o controle transacional sobre as operações feitas no banco de dados.

O servidor de aplicações *web* Glassfish, apesar de não ser indicada como a melhor escolha pelas comunidades de desenvolvedores, em decorrência do anúncio da Oracle de descontinuar a sua versão Enterprise, ainda na versão 4.0 ele possuía maior usabilidade que o TomEE, que exigia excessivas configurações para integrar as tecnologias citadas. O WildFly seria uma alternativa recomendada pela própria Oracle, mas ainda não era suportada pelo Netbeans 7.4 e o projeto ainda não estava em versão final.

O Gerenciador de projetos Maven foi importante no desenvolvimento do projeto pois afastou do programador a preocupação de baixar, instalar, configurar e gerenciar os pacotes java de componentes externos, assim como suas dependências, deixando o projeto “portável”, em que apenas os códigos-fonte precisariam ser distribuídos, deixando para o Maven o papel de gerenciar os pacotes java do qual o projeto depende.

Estas tecnologias uma vez integradas resultaram no aumento da produtividade e abstração das classes da aplicação.

5 CONCLUSÃO

O objetivo principal deste trabalho foi a implementação uma fábrica de software definida a partir de processos. O sistema foi modelado tendo como base uma determinada disciplina de um curso específico. Contudo, a solução se aplica para qualquer disciplina acadêmica que seja realizada visando simular uma fábrica de software e mesmo para fábricas que desenvolvem projetos de software sob demanda ou para atender nichos de mercado.

O referencial teórico abrangeu a fundamentação conceitual para definir fábrica de software a partir de processos. Portanto, referências relacionadas a conceitos de fábricas de software, processos de software e modelos de qualidade foram levantadas e seus conteúdos considerados na construção do texto.

O aplicativo desenvolvido é para *web*. Assim, conceitos sobre desenvolvimento para *web* que utilizam tecnologias que permitem caracterizar as aplicações como ricas foram considerados. Tendo em vista que o aplicativo desenvolvido será utilizado por uma turma de alunos organizados em equipes e que cada equipe desenvolve um módulo (da definição dos requisitos aos testes de integração) é importante que o sistema seja de uso fácil. Desta forma, os recursos de interação (interface) oferecidos pelas tecnologias utilizadas juntamente com a linguagem Java foram relevantes para a implementação do sistema.

Em relação às tecnologias, pode-se destacar que a biblioteca de componentes para JSF Primefaces possui em seu site um *showcase* que é uma demonstração do uso de componentes, contendo um exemplo básico da construção de uma página HTML e a interação com a classe ManagedBean. Isso facilita muito o entendimento inicial de funcionamento e uso dessa tecnologia. No próprio site há uma grande quantidade de componentes disponíveis e eles são de fácil utilização.

O *Framework JavaServer Faces* (JSF) permite que as páginas *web* sejam facilmente construídas usando componentes, que serão interpretados pelo servidor de aplicação e disponibilizados em forma de HTML, CSS e JavaScript para o usuário final. Sendo o *PrimeFaces* uma alternativa de pacote de componentes que estendem ainda mais o uso do JSF, deixando a interface de uso rica e elegante.

O uso do Framework Hibernate permite que a persistência dos dados para o servidor seja realizada de forma natural para a programação orientada a objetos, não exigindo que o programador tenha conhecimento profundo de SQL. A persistência é controlada pela especificação JTA (*Java Transaction API*) do Glassfish que gerencia o fluxo de transações do

banco de dados. Pela especificação JPA (*Java Persistence API*) é possível fazer engenharia reversa do banco de dados gerando as classes de entidade de forma automática, diminuindo o esforço para criação de novas partes do sistema. O CDI (*Contexts and Dependency Injection*), por sua vez, permite a inversão de dependência das classes DAO (*Data Access Object*) para a interface, facilitando a interpretação do código e aumentando a produtividade com o uso de anotações.

O uso do Hibernate permitiu a criação da modelagem inicial do sistema sem o uso expressões SQL. Tornando mais fácil seu desenvolvimento. Basta apenas criar a classe de entidade e mapeá-la no arquivo de configuração do Hibernate para que se possa fazer operações na tabela do banco.

Destaca-se, ainda, o gerenciamento realizado pelo Maven, que compila e implanta o aplicativo sem que o programador se preocupe em ficar baixando os pacotes Java das quais o projeto depende.

Em termos de implementação das funcionalidades do sistema, as essenciais foram implementadas. Todos os cadastros que são necessários para armazenar o que é produzido durante o ciclo de vida estão prontos, também é possível compor equipes e projetos. O sistema permite o gerenciamento das suas funcionalidades básicas.

Como sequência do trabalho está a geração de relatórios de acompanhamento pelo professor e pelas próprias equipes. Além da gerência de configuração e de métricas e de outras funcionalidades necessárias ao gerenciamento de uma fábrica de software visando atender modelos de qualidade.

REFERÊNCIAS

ANAYA, Victor; ORTIZ, Angel, **How enterprise architectures can support integration**, First International Workshop on Interoperability of Heterogeneous Information Systems, 2005, p. 25-30.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 12207**: **tecnologia de informação - processos de ciclo de vida de software.** Rio de Janeiro: 1998.

BARRETO JUNIOR, Celso Gomes. **Agregando frameworks de infra-estrutura em uma arquitetura baseada em componentes: um estudo de caso no ambiente AulaNet**. Dissertação de Mestrado. Programa de Pós-Graduação em Informática da PUC-Rio. 2006. Disponível em: <https://docs.google.com/gview?url=http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0410823_06>. Acesso em: 15 mai. 2013.

BERTOLLO, Gleidson; FALBO, Ricardo A. **Apoio automatizado à definição de processos de software em níveis**. In: II Simpósio Brasileiro de Qualidade de Software (SBQS'2003), 2003. p. 77-91.

BLAHA, Michael; JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2**. 2ª ed. Rio de Janeiro: Elsevier, 2006.

BUCHNER, Björn; BÖTTCHER, Axel; STORCH, Christian. **Evaluation of java-based open source web frameworks with ajax support**. 14th IEEE International Symposium on Web Systems Evolution, 2012, p. 45-49.

BUSCHMANN, Frank, MEUNIER, Regine, ROHNERT, Hans, SOMMERLAD, Peter; STAL, Michael **Pattern-oriented software architecture: A system of patterns.** Chichester, UK: John Wiley & Sons, 1996.

CELEPAR. Protocolo HTTP (Hypertext Transfer Protocol). Bate Byte. 2009. Disponível em: http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=138. Acesso em: 17 dez. 2013.

DEMIR, Ahmet, **Comparison of model-driven architecture and software factories in the context of model-driven development**. In: Fourth and Third International Workshop on Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software (MBD/MOMPES 2006), 2006, p. 75- 83.

FABRI, José Augusto; L'ERÁRIO, Alexandre; TRINDADE, André Luiz Presende; PESSÔA, Marcelo S. de Paula; SPÍNOLA, Mauro de Mesquita. **Desenvolvimento e replicação de uma fábrica de software**, VI Simpósio Internacional de Melhoria de Processos de Software, 2004, p. 1-50.

FAYAD, Mohamed E. **Introduction to the computing surveys' electronic symposium on object-oriented application frameworks**. ACM Computing Surveys, v. 32, n. 1, March 2000, p. 1-9.

FAYAD, Mohamed E.; SCHIMIDT, Douglas C.; Johnson, Ralph E. **Building application frameworks: object-oriented foundations of framework design**. New York: J. Wiley, 1999.

FERNANDES, Agnaldo A.; TEIXEIRA, Descartes S. **Fábrica de software: implantação e gestão de operações**. São Paulo: Atlas, 2004.

FIORINI, Soeli T., VON STAA, Arndt, BATISTA, Renan B. **Engenharia de Software com CMM**. Brasport, Rio de Janeiro, 1998.

FRANKEL, David S. **Business process platforms and software factories**. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'05). International Workshop on Software Factories, 2005, p. 1-5.

FUGGETTA, Alfonso. **Software process: a roadmap**. In: The Future of Software Engineering (ICSE'2000), ACM Press, 2000, p. 25-34.

GREENFIELD, Jack; SHORT, Keith. **Software factories assembling applications with patterns, models, frameworks and tools**. Conference on Object Oriented Programming Systems Languages and Applications, 2003, p.16-27.

IBM. **JavaServer Faces life cycle**. Disponível em: <http://publib.boulder.ibm.com/infocenter/iadthelp/v6r0/index.jsp?topic=/com.ibm.etools.webtoolscore.doc/topics/cjsflifecycle.html>. Acesso em: 16 mai. 2013

ICEFACES. **Icefaces**. Dispponível em: <http://www.icesoft.org>. Acesso em: 13 set. 2013. object-oriented application frameworks. ACM Comput. Surv., 32(1):1–9.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 9000:2000, Sistemas de gestão da qualidade – Fundamentos e vocabulário**, 2000.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC 15504 **International Standard Organization. ISO/IEC TR 15504 - Software process**. Montreal: ISO/IEC JTC1 SC7, 1999.

MANGAN Peter J.; SADIQ, Shazia. **On building workflow models for flexible processes**. In: Thirteenth Australasian Database Conference (ADC2002), v.5, 2002, p. 1-7.

OKANOVIC, Vansada. **Designing a web application framework**. In: 18th International Conference on Systems, Signals and Image Processing (IWSSIP), 2011, p. 1-4.

ORACLE. **JavaServer Faces technology overview**. Disponível em: <http://www.oracle.com/technetwork/java/javaee/overview-140548.html>. Acesso em 16 mai. 2013.

PEREIRA, Adriano; COGO, Vinícius Vielmo; CHARÃO, Andrea Schwertner. **Frameworks para desenvolvimento rápido de aplicações web: um estudo de caso com CakePHP e Django**. 10º Fórum Internacional de Software Livre (FISL 10). Disponível em: http://w3.ufsm.br/pet-cc/?p=766. Acesso em: 15 mai. 2013.

PRESSMAN, Roger. **Engenharia de software**. McGraw-Hill: São Paulo, 2008.

PRIMEFACES. **PrimeFaces**. Disponível em: <http://primefaces.org/>. Acesso em: 13 set. 2013.

RICHFACES. **Richfaces.** Disponível em: <http://www.jboss.org/richfaces>. Acesso em: 13 set. 2013.

ROCHA, Ana Regina C., MALDONADO, João Carlos, WEBER, Kival C. **Qualidade de software**. São Paulo: Prentice Hall, 2001.

ROGÉRIO, Pedro. **CSS Frameworks**. 2008. Disponível em http://www.cssnolanche.com.br/css-frameworks/. Acesso em 15 de maio de 2013.

SENGER, Yara. MAGALHÃES, Eder. **JSF2 tirando proveito dos componentes: RichFaces / IceFaces / PrimeFaces**. Disponível em: <http://www.slideshare.net/edermag/jsf2-tirando-prove>. Acesso em: 15 mai. 2013.

SOFTEX. **MPS.BR – Melhoria de Processo do Software Brasileiro – Guia Geral**. Brasília: Sociedade Softex, 2007.

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro. Guia de Implementação – Parte 2: Fundamentação para Implementação do Nível F do MR-MPS**, Brasília: Sociedade Softex, 2011.

SOFTWARE ENGENEERING INSTITUTE. **CMMI for development**, Pittsburgh: Software Engineering Institute, 2006.

SZYPERSKI, Clemens. **Component software: beyond object-oriented programming**, Addison-Wesley, 1997.

TAYLOR, David A. **Engenharia de negócios com tecnologia de objetos.** Rio de Janeiro: Axcel Books, 2003.

WEBER, Sergio; HAUCK, Jean C. R.; WANGENHEIM, Christiane G. V. **Estabelecendo processos de software em micro e pequenas empresas**. In: SBQS – Simpósio Brasileiro de Qualidade de Software, Porto Alegre: p. 1-16, 2005.

WFMC TC-1011. **Workflow management coalition. Terminology and glossary**, Document Number WFMC-TC-1011, version 2.0, 1996, p.1-58.