UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FURBUP: UM PROCESSO DE SOFTWARE PARA USO ACADÊMICO BASEADO NO OPENUP

JOÃO PAULO PEDRI

BLUMENAU
2008

2008/1-19

JOÃO PAULO PEDRI

FURBUP: UM PROCESSO DE SOFTWARE PARA USO ACADÊMICO BASEADO NO OPENUP

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Everaldo Artur Grahl, Mestre - Orientador

BLUMENAU
2008

2008/1-19

FURBUP: UM PROCESSO DE SOFTWARE PARA USO ACADÊMICO BASEADO NO OPENUP

Por

JOÃO PAULO PEDRI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB
Membro:	Prof. Adilson Vahldick, Especialista – FURB
Membro:	Prof. Marcel Hugo, M.Eng. – FURB

Blumenau, 01 de julho de 2008

Dedico este trabalho a todos que de alguma forma me incentivaram e apoiaram durante a sua elaboração, especialmente aos meus pais, avós e à minha namorada Himene.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e força para superar os desafios.

À minha família, que sempre me incentivou e apoiou. Especialmente aos meus pais e avós que me acompanharam em toda a jornada.

À minha namorada, pela compreensão em função de todo tempo dedicado a elaboração do trabalho.

Aos meus amigos, pela compreensão de minha ausência em função do tempo dedicado ao trabalho. Especialmente aos amigos e companheiros de jornada na faculdade Éverton Demo e Leandro Beszczynski, pelo apoio aos trabalhos realizados ao longo do curso.

Ao meu cunhado Rafael pela força no desenvolvimento deste trabalho.

Ao meu orientador, Everaldo Artur Grahl, por ter acreditado na conclusão deste trabalho.

Aos professores que direta ou indiretamente contribuíram para minha formação.

Os que se encantam com a prática sem a ciência são como os timoneiros que entram no navio sem timão nem bússola, nunca tendo certeza do seu destino.

Leonardo da Vinci

RESUMO

Este trabalho visa apresentar o FurbUP, processo de desenvolvimento de software, elaborado para utilização no ambiente acadêmico da Universidade Regional de Blumenau (FURB). Esse processo foi concebido para fornecer aos acadêmicos dos cursos de ciências da computação e sistemas de informação, uma alternativa viável para o desenvolvimento de atividades relacionadas às disciplinas de engenharia de software. Com objetivo de validar o processo, será apresentada a especificação e implementação de um estudo de caso, representado através de um sistema de reserva de laboratórios. Junto a este estudo de caso, será disponibilizado aos alunos um *template* com sua especificação, com o intuito de auxiliá-los no desenvolvimento de um produto de software. O FurbUP é totalmente baseado no *Open Unified Process* (OpenUP), processo de software *open source* desenvolvido pela International Business Machine (IBM).

Palavras-chave: Processo de software. Métodos de software. OpenUP.

ABSTRACT

This paper aims to present the FurbUP, a software development process, elaborated to be used at the Universidade Regional de Blumenau (FURB) academic environment. This process was conceived to provide a viable alternative for activities related to the software engineering disciplines, for the academics of the computer science and information technology courses. With the purpose of validating the process, the specification and implementation of a case study will be presented, depicted through a laboratory booking system. Together with this case study, it will be made available a template with its specification, in order to help them in the development of a software product. The FurbUP is totally based on the Open Unified Process (OpenUP), an open source software developed by the International Business Machine (IBM).

Key-words: Software process. Software methods. OpenUP.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AM – *Agile Modeling*

CASE – *Computer Aided Software Engineering*

EA – Enterprise Architect

EF – Eclipse Foundation

EPFC – Eclipse Process Framework Composer

ERS – Engenharia de Requisitos de Software

ES – Engenharia de Software

FACEB – Faculdade Cenecista de Brasília

FURB – Universidade Regional de Blumenau

IBM – International Business Machine

JAD – *Joint Application Design*

MD – MetoDes

MVC – *Model, View* e *Controller*

OpenUP – *Open Unified Process*

PID – ProcessID

RUP – *Rational Unified Process*

UFCG – Universidade Federal de Campina Grande

UCP – *Use Case Points*

UMA – *Unified Method Architecture*

UML – *Unified Modeling Language*

XP – *eXtreme Programming*

YP – easYProcess

SUMÁRIO

1 INTRODUÇÃO

O crescimento da produção de software, devido a grande variedade de produtos de software, é um dos principais responsáveis pela evolução e utilização da disciplina de Engenharia de Software (ES). Com isso, houve a necessidade da utilização de um processo de desenvolvimento. Conforme Sommerville (2003, p. 7), "Um processo de software é um conjunto de atividades e resultados associados que geram um produto de software." A produção de um software envolve fundamentos teóricos e práticos, introdução prévia a ES e a utilização de um processo de desenvolvimento, assim como de tecnologias aceitas pelo mercado.

A universidade é o lugar ideal para que a teoria seja aliada à prática com o objetivo de ensinar ao estudante um conceito real relativo aos trabalhos desenvolvidos na ES. Para isso é necessário aplicar os conhecimentos adquiridos no estudo das disciplinas de ES na elaboração de um produto de software. O resultado esperado é alcançado, quando a escolha do processo a ser utilizado é adequada ao cenário da instituição.

Sommerville (2003, p. 7-8) afirma que "Se um processo inadequado for utilizado, isso provavelmente reduzirá a qualidade ou a utilidade do produto de software a ser desenvolvido." Sendo assim, alguns processos são mais adequados do que outros para certos tipos de aplicação.

Existe uma grande variedade de processos, entre eles o *Rational Unified Process*[1] (RUP), *eXtreme Programming*[2] (XP) e o *Open Unified Process* (OpenUP). Cada processo ostenta suas características e abordagens de desenvolvimento. O RUP oferece uma abordagem baseada em disciplinas para conceder tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é a produção de software com alta qualidade que atenda às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis. O XP é uma metodologia ágil voltada para equipes pequenas e médias que desenvolvem software fundamentado em requisitos instáveis e que se alteram constantemente. Sua meta é o

[1] RUP é um processo de desenvolvimento de software influenciado pelo processo unificado e faz uso extensivo da *Unified Modeling Language* (UML) (SCOTT, 2003, p. 19).

[2] XP é um processo de software (ou metodologia de desenvolvimento de software) que possui o foco no método ágil de desenvolvimento. Segundo Kent Beck, criador da XP, a *eXtreme Programming* é uma metodologia ágil para pequenas e médias equipes de desenvolvimento de software, trabalhando com requisitos vagos e em constante mudança (ASTELS; MILLER; NOVAK, 2002, p. XXIV).

desenvolvimento rápido de um projeto de software e visa garantir, assim como o RUP, a satisfação dos usuários e o cumprimento das estimativas propostas. O OpenUP é um processo *open source*, focado apenas no conteúdo fundamental necessário, disponibilizando um conjunto simplificado de ações, seguindo o método iterativo e incremental de desenvolvimento. Entre outras características, o OpenUP é extensível, ou seja, pode ser utilizado de acordo com as necessidades do projeto.

Para atingir o sucesso em um projeto de software no ambiente acadêmico, obtendo um produto de qualidade e entregue no prazo previsto pelas disciplinas, é necessário o uso de um processo de desenvolvimento prático, porém completo. Nas disciplinas de ES, da Universidade Regional de Blumenau (FURB), não existe um roteiro didático para auxiliar os acadêmicos no desenvolvimento de seus projetos de software. Além disso, apesar de boa vontade por parte dos professores, a integração entre as disciplinas ainda é pequena.

Em busca de uma solução adequada às necessidades acadêmicas das disciplinas de ES da FURB, surgiu a oportunidade de elaborar um processo de desenvolvimento de software chamado FurbUP, utilizando como base o processo de desenvolvimento OpenUP. Junto a este novo processo, foi desenvolvido um *template*[3] para auxiliar os acadêmicos no desenvolvimento de aplicações. Também foi disponibilizado um estudo de caso, representado através de um sistema de reserva de laboratórios, que ilustra a criação de vários artefatos.

Para disponibilizar o processo FurbUP foi utilizada a ferramenta Eclipse Process Framework Composer (EPFC) que tem como objetivo auxiliar a autoria de métodos, processos, gestão de configuração e publicação de processos.

A elaboração de um processo de software, baseado no modelo OpenUP, pode ser considerado significativo, pois apesar dos métodos utilizados nas disciplinas de ES já serem vastamente estudados, o OpenUP reúne as principais características do desenvolvimento ágil apresentado no XP e do desenvolvimento em iterações do RUP, além de ser um modelo novo, *open source* e atualmente pouco explorado pelas instituições acadêmicas. Esta combinação de características permite uma maior qualidade dos resultados gerados pela equipe de desenvolvimento. Atualmente, não existe publicação de um texto completo apresentando as principais características do processo OpenUP em português, sendo assim, os conteúdos essenciais para a concepção do FurbUP foram traduzidos.

Foram mapeados os temas que envolvem as disciplinas da área de ES da FURB para

[3] É um documento formatado que o acadêmico irá preencher com as informações do projeto em construção.

que se tenha através do FurbUP, um processo que favoreça o trabalho em equipe e que integre os professores das respectivas disciplinas envolvidas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar um processo de software para ser aplicado em disciplinas de ES na FURB, utilizando o processo OpenUP.

Os objetivos específicos do trabalho são:

a) traduzir as principais características do processo OpenUP para a língua portuguesa;
b) publicar o processo FurbUP utilizando a ferramenta EPFC;
c) implementar um sistema de reserva de laboratórios que será usado para aplicar o processo FurbUP;
d) gerar um *template* para auxiliar o desenvolvimento de aplicações utilizando o FurbUP.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em quatro capítulos. No segundo capítulo é apresentada a fundamentação teórica utilizada para o desenvolvimento do trabalho. Nele são discutidos a importância de um processo de software. Também são examinadas as principais características do processo de software OpenUP e apresentados conceitos sobre a ferramenta EPFC. São evidenciadas as características das disciplinas de ES da FURB. O capítulo traz ainda uma descrição de alguns trabalhos correlatos.

No terceiro capítulo é apresentado o desenvolvimento, iniciando com a tradução das características essenciais do OpenUP, seguido da publicação e elaboração do processo FurbUP baseado nos conteúdos traduzidos, da especificação do estudo de caso representado através de um sistema de reserva de laboratórios e da elaboração do *template*. Finalizando, no quarto capítulo, são apresentadas as conclusões e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os aspectos teóricos relevantes à elaboração do presente trabalho. É realizado um estudo sobre a importância de um processo de software. São analisados os modelos de processo de software, assim como algumas vantagens e desvantagens de tais modelos. São também examinadas as principais características do processo de software OpenUP, sendo elencadas as fases do seu ciclo de vida. É apresentada uma visão geral da ferramenta EPFC, descrevendo sua estrutura através de conteúdos de métodos e funções específicas. São evidenciadas as características das disciplinas de ES da FURB, dando-se destaque a importância de tais disciplinas no aprendizado. Por último, são expostas algumas características dos trabalhos correlatos que serviram de base, assim como o processo OpenUP, para a publicação do processo proposto.

2.1 PROCESSOS DE SOFTWARE

Conforme Paula Filho (2003, p. 11), um processo é definido quando possui documentações que detalham o que foi realizado (produto), quando (passos), por quem (pessoa que pratica a ação), as coisas que são utilizadas (entrada) e as coisas que são produzidas (resultados). Os processos em geral podem ser contemplados com mais ou menos detalhes. Processo de desenvolvimento de software está diretamente relacionado ao ciclo de vida do software, pois descreve a evolução do produto de software desde a concepção até a implementação, entrega, utilização e manutenção.

Na engenharia de software, processos são utilizados para procedimentos como desenvolvimento, manutenção, aquisição e contratação de software. Pode-se também definir sub-processos para cada um desses citados anteriormente.

Conforme Sommerville (2003, p. 36), existem muitos processos de software diferentes, mas existem ações comuns entre eles, os quais:

a) especificação das funcionalidades de software;
b) projeto e implementação de software, para que na produção de software sejam atendidas todas as suas especificações;

c) validação do software para que atenda o que o cliente deseja;

d) progresso de software para atender as necessidades suscetíveis do cliente.

Paula Filho (2003, p. 12) afirma que "Em um processo de desenvolvimento de software, o ponto de partida para a arquitetura de um processo é a escolha de um modelo de ciclo de vida."

O ciclo de vida de um projeto de software, utilizando um processo de desenvolvimento, define as fases que conectam o início e o fim do mesmo. Um ciclo de vida geralmente define qual o trabalho técnico deve ser utilizado em cada fase, quando as entregas devem ser geradas em cada fase, o que entregar em cada fase e como cada entrega é revisada, verificada e validada. Também é importante saber quem está envolvido em cada fase e como controlar e aprovar cada fase.

2.2 MODELOS DE PROCESSOS DE SOFTWARE

Sommerville (2003, p. 8) afirma que "Um modelo de processo de software é uma descrição simplificada de um processo de software, que é apresentada a partir de uma perspectiva específica.". O que mais se destaca em um modelo de processo são as atividades que fazem parte de um processo de software, produtos de software e o papel dos envolvidos na engenharia de software.

Como exemplos dos diferentes tipos de modelos de processo de software que podem ser desenvolvidos são:

a) **modelo de *workflow*[4]**, que apresenta a seqüência de atividades no processo, com suas entradas, saídas e dependências;

b) **modelo de fluxo de atividades**, que reproduz o processo como um conjunto de atividades, a qual cada uma realiza modificações nos produtos de trabalho gerados. Esse modelo apresenta como uma entrada para o processo, tal como uma definição de escopo de um sistema, é transformada em uma saída;

c) **modelo de papel ou ação**, reproduz os papéis das pessoas envolvidas no processo de software, assim como as atividades que elas são responsáveis.

---

[4] É um termo em inglês para designar um processo de negócio. Seu uso neste idioma estendeu-se a tudo aquilo vinculado a ferramentas de informática que contribuem à automatização e ao controle de processos.

Cada modelo de processo representa um processo sob aspecto particular, de uma maneira que mostre apenas informações parciais sobre o processo.

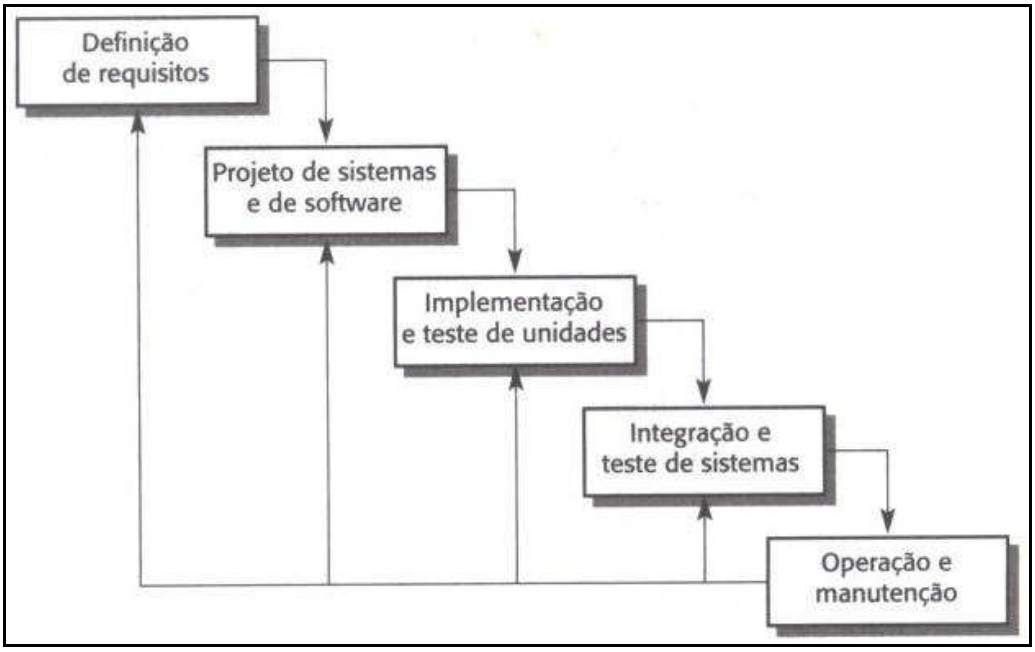
Sommerville (2003, p. 37) apresenta diferentes modelos de processo para explicar diferentes abordagens do desenvolvimento de software, os quais são:

a) modelo em cascata;
b) desenvolvimento evolucionário;
c) desenvolvimento formal de sistemas;
d) desenvolvimento orientado a reuso;
e) desenvolvimento incremental;
f) desenvolvimento em espiral.

2.2.1 Modelo em cascata

Esse modelo é formado pelas atividades de especificação, desenvolvimento, validação e evolução, que são importantes ao processo, e as considera como fases divididas do processo, como a definição dos requisitos, o projeto de software, a implementação e teste de unidade, a integração e testes de sistemas e operação e manutenção.

Desde sua publicação tem sido muito utilizado para o desenvolvimento de sistemas práticos. É um modelo em que se deve seguir uma seqüência entre fases, por isso foi atribuído ao mesmo o nome cascata ou ciclo de vida de software. As fases essenciais do modelo definem as atividades de desenvolvimento, como ilustra a Figura 1.



Fonte: Sommerville (2003, p. 38).

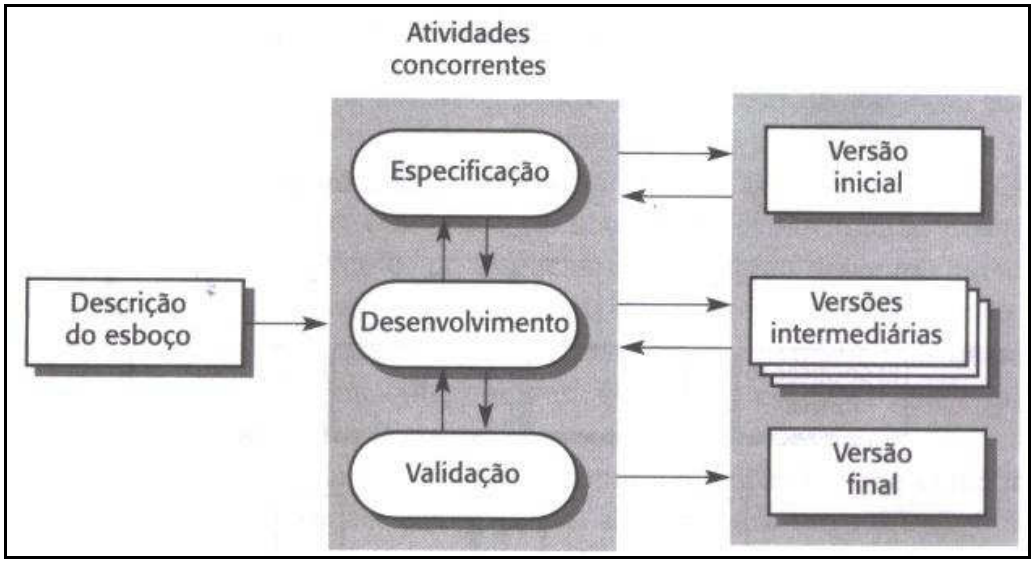
Figura 1 – Modelo em cascata

Cada fase tem como resultado uma documentação padrão que deve ser aprovada e assinada para que se inicie a fase posterior, ou seja, a fase seguinte não deve se iniciar até que a fase precedente tenha sido totalmente encerrada.

O problema do modelo em cascata é sua inflexível divisão do projeto em fases separadas, o que dificulta possíveis alterações no desenvolvimento de um projeto. É um modelo que deve ser usado somente quando os requisitos forem bem entendidos.

2.2.2 Desenvolvimento evolucionário

O desenvolvimento evolucionário tem como base desenvolver e implementar um produto inicial, que é submetido aos comentários e críticas do usuário. O produto vai sendo aprimorado através de inúmeras versões, até que o produto de software esperado tenha sido desenvolvido, conforme ilustra a Figura 2. As atividades de especificação, desenvolvimento e validação são realizadas paralelamente, com um eficiente *feedback* entre elas (SOMMERVILLE, 2003, p. 39).



Fonte: Sommerville (2003, p. 39).

Figura 2 – Desenvolvimento evolucionário

Existem dois tipos de desenvolvimento evolucionário, os quais são:

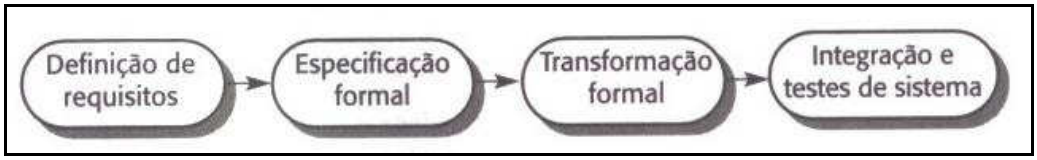
a) desenvolvimento exploratório, em que o objetivo do processo é trabalhar junto do usuário para descobrir e explorar seus requisitos, de maneira incremental, até que o sistema final seja entregue. O sistema evolui quando novas características são atribuídas à medida que elas são propostas pelo usuário;
b) fazer protótipos descartáveis, em que o objetivo do desenvolvimento evolucionário é entender os requisitos do usuário e alcançar uma melhor definição dos requisitos do sistema. O protótipo é usado para fazer experimentos com os requisitos do usuário que não estejam bem entendidos.

O problema do desenvolvimento evolucionário é que o processo não é visível, pois o desenvolvimento ocorre de maneira rápida, não sendo viável produzir documentos que retratem cada versão do produto de software. Com isso mudanças constantes durante o desenvolvimento tendem a abalar a estrutura do software.

2.2.3 Desenvolvimento formal de sistemas

O desenvolvimento formal de sistemas é uma abordagem do desenvolvimento de software que tem princípios comuns ao do modelo em cascata, mas cujo processo tem como fundamento uma transformação matemática formal de uma especificação de um sistema em

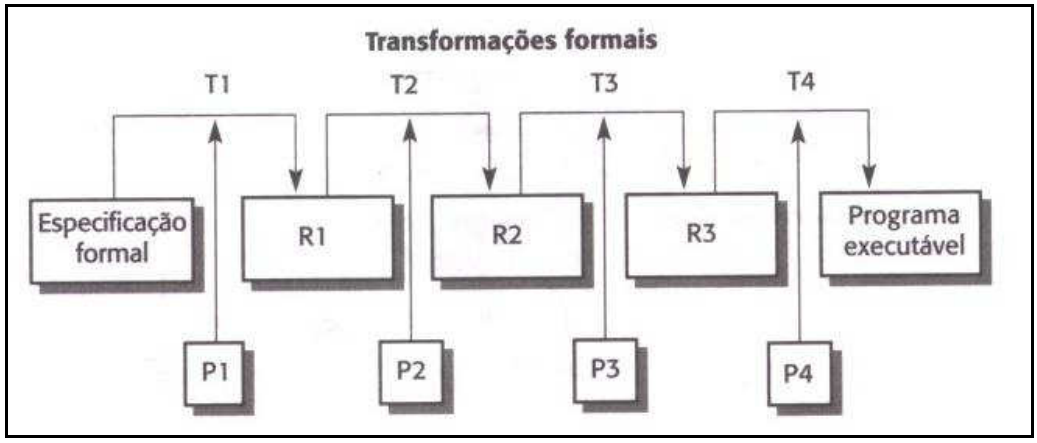
um programa executável, conforme ilustra a Figura 3 (SOMMERVILLE, 2003, p. 40).



Fonte: Sommerville (2003, p. 40).

Figura 3 – Desenvolvimento formal de sistemas

Neste modelo a especificação de requisitos é definida novamente em uma especificação formal delinear, que é representada em uma notação matemática. Os processos de desenvolvimento de projeto, implementação e teste de unidade são trocados por um processo transformacional, em que a especificação formal é aprimorada através de inúmeras transformações, em um programa, conforme ilustra a Figura 4.



Fonte: Sommerville (2003, p. 40).

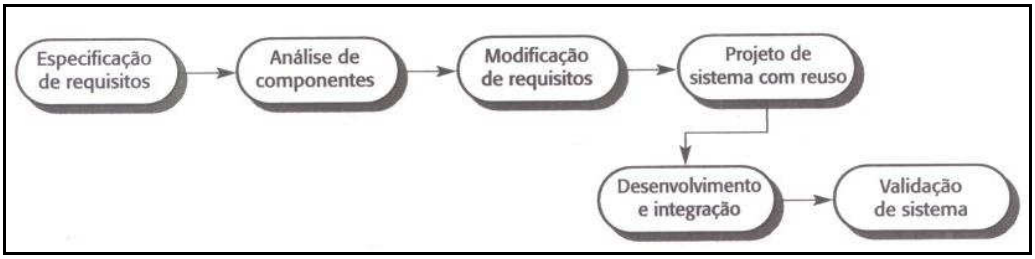
Figura 4 – Transformação formal

Os processos com base em transformação formal não são largamente utilizados, pois requerem um conhecimento especializado, e conforme Sommerville (2003, p. 41), “para a maioria dos sistemas, esse processo não oferece vantagens significativas de custo ou qualidade em relação a outras abordagens.”

2.2.4 Desenvolvimento orientado a reuso

Na maioria dos projetos de software, acontece algum tipo de reuso de software. Isso ocorre quando pessoas que fazem parte da equipe do projeto conhecem códigos ou projetos que possuem a mesma função ou efeito ao exigido. Ao encontrar essas similaridades, são realizadas as modificações conforme a necessidade do projeto e reunido no sistema. O modelo

comum de processo para desenvolvimento orientado a reuso é ilustrado na Figura 5.



Fonte: Sommerville (2003, p. 42).

Figura 5 – Desenvolvimento orientado a reuso

Conforme Sommerville (2003, p. 42), embora os estágios de especificação de requisitos e validação sejam comparáveis com outros processos, os estágios intermediários no processo orientado a reuso são diferentes, os quais são:

a) análise de componentes;
b) modificação de requisitos;
c) projeto de sistema com reuso;
d) desenvolvimento e integração.

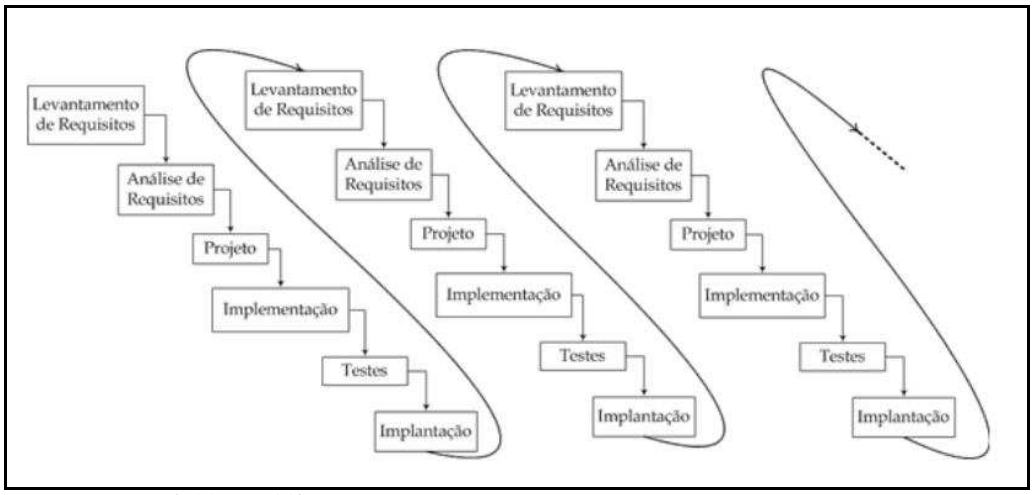
A vantagem deste modelo de processo é reduzir consideravelmente a quantidade de software a ser desenvolvida, diminuindo custos e riscos, obtendo rapidamente a entrega do software.

2.2.5 Desenvolvimento incremental

No desenvolvimento incremental, o sistema é dividido em subsistemas por funcionalidades, ou seja, o desenvolvimento de um produto de software é dividido em ciclos. Em cada ciclo de desenvolvimento podem ser identificados os estágios de análise, projeto, implementação e testes. Essa característica é diferente da adotada no modelo em cascata, na qual as fases de análise, projeto, implementação e testes são realizadas uma única vez.

Em um processo de desenvolvimento incremental, os usuários identificam, em uma delineação inicial, as funções a serem proporcionadas pelo sistema. Em seguida, conforme Sommerville (2003, p. 43), “é definida uma série de estágios de entrega, com cada estágio fornecendo um subconjunto das funcionalidades do sistema.”. Assim, utilizando o modelo incremental, o desenvolvimento evolui em versões até que o sistema completo esteja construído. É importante observar que apenas parte dos requisitos é considerada em cada ciclo

de desenvolvimento. O modelo de desenvolvimento incremental pode ser visto como uma generalização do modelo em cascata, conforme ilustra a Figura 6.



Fonte: Bezerra (2002, p. 33).

Figura 6 – No processo incremental e iterativo, cada iteração é uma mini-cascata

Existem alguns problemas com o desenvolvimento incremental. As iterações devem ser relativamente pequenas e cada iteração deve produzir alguma funcionalidade para o sistema. Portanto, pode ser custoso mapear os requisitos dos clientes dentro de iterações de tamanho correto. Como os requisitos não são definidos em detalhes até que uma iteração seja implementada, é árduo identificar aptidões comuns que todas as iterações exijam.

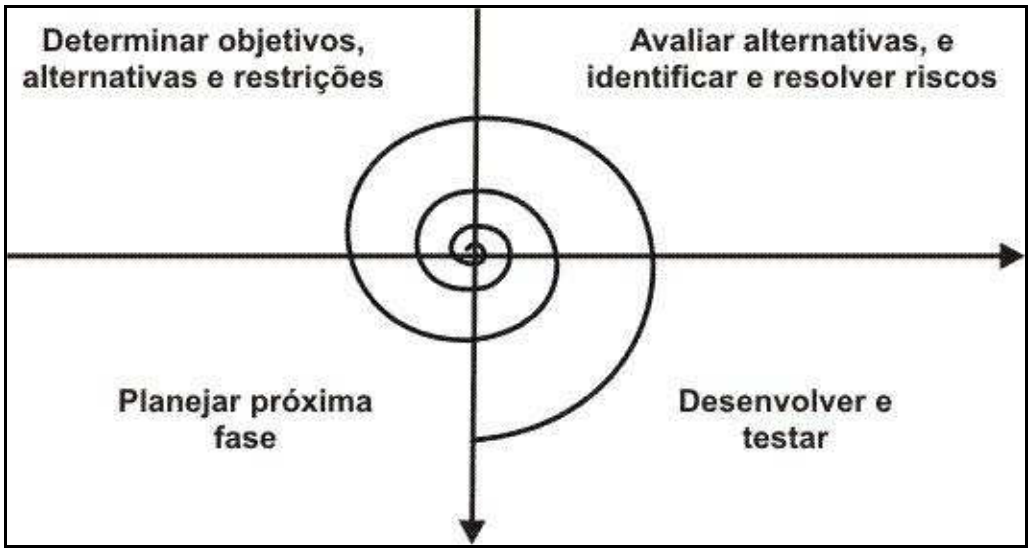
2.2.6 Desenvolvimento em espiral

O modelo de processo de desenvolvimento em espiral engloba as melhores práticas dos modelos em cascata e evolucionário. Ao invés de representar um processo de software com uma seqüência de atividades com algum retorno entre uma tarefa e outra, o processo é representado como uma espiral, conforme ilustra a Figura 7. Adiciona-se um novo elemento chamado análise de risco, o qual circunstâncias adversas podem atrapalhar o processo de desenvolvimento e a qualidade do produto de software. Cada ciclo na espiral representa uma fase do processo de desenvolvimento de software. Assim, o ciclo mais interno pode estar relacionado com o estudo de viabilidade e com operacionalidade do sistema. O próximo ciclo com a definição dos requisitos e o próximo com o projeto do sistema e assim por diante. Não existem fases fixas.

Conforme Sommerville (2003, p. 45), cada ciclo da espiral é dividido em quatro

setores, os quais são:

a) definição de objetivos;
b) avaliação e redução de riscos;
c) desenvolvimento e validação;
d) planejamento.



Fonte: adaptado de Sommerville (2003, p. 45).

Figura 7 – Desenvolvimento em espiral

Este modelo prevê a eliminação de problemas de alto risco através de um planejamento e projetos cuidadosos. As atividades podem ser ordenadas como uma espiral que tem muitos ciclos. Possibilita ao desenvolvedor entender e reagir aos riscos em cada ciclo.

2.3 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE OPENUP

Projetos diferentes têm diferentes necessidades de processo. Fatores típicos ditam as necessidades para um processo mais formal ou ágil, tais como tamanho de equipe e localização, complexidade da arquitetura, novidades tecnológicas, conformidade com normas, entre outros. Apesar disso, existem boas práticas de desenvolvimento de software que beneficiam qualquer equipe de projetos e ajudam elas a serem mais eficazes.

O OpenUP é um processo de desenvolvimento de software *open source* que tem o objetivo de cobrir um grande conjunto de necessidades de desenvolvimento. Contém o conjunto mínimo de práticas que ajudam as equipes a serem mais eficazes no desenvolvimento de software. Além disso, traz como característica o destaque ágil ao

processo de desenvolvimento, concentrado apenas nos conteúdos essenciais, disponibilizando um conjunto reduzido de produtos de trabalho, papéis, tarefas e guias (PAPO, 2007, p. 29). Assim, ele não proporciona orientação para diversos tópicos com os quais os projetos possam lidar, tais como equipes de tamanho grande, conformidade, situações contratuais, aplicações de segurança ou de missão crítica, orientação para tecnologia específica, etc. Entretanto, o OpenUP é completo no sentido de que ele pode ser manifestado como um processo completo para construir um sistema. Para abordar necessidades que não sejam cobertas neste conteúdo, o OpenUP é extensível para ser utilizado como uma base na qual conteúdos de processo possam ser adicionados ou personalizados conforme seja necessário. Para a elaboração do OpenUP foi utilizado o EPFC, que permite construir processos e *frameworks*[5] de processos de desenvolvimento.

O OpenUP foi desenvolvido pela International Business Machine (IBM) com base nos processos RUP e XP, a fim de juntar as melhores qualidades de cada um desses métodos. A utilização do OpenUP torna-se muito adequada para instituições, pois os elementos que o compõem são flexíveis a extensões. Recentemente, o OpenUP foi doado pela IBM para a Eclipse Foundation (EF), a qual disponibiliza a ferramenta EPFC para construção de novos *frameworks*.

Papo (2007, p. 29) acrescenta que o OpenUP é um processo de software completo, que possui práticas de desenvolvimento incremental e que é customizável de acordo com as necessidades de um determinado projeto.

O OpenUp é um processo ágil. Apesar de o OpenUp ser leve, existe muito mais para ser ágil do que simplesmente ser leve. As mais reconhecidas práticas ágeis são intencionadas a conseguir uma equipe comunicando-se entre si, proporcionando uma compreensão compartilhada do projeto. Métodos ágeis têm chamado a atenção para a importância da compreensão da coordenação, beneficiando os *stakeholders*[6] no que diz respeito a entregas improdutivas e a formalidades.

O processo OpenUP tem a característica de utilização de métodos ágeis definidos como:

---

[5] No desenvolvimento de software, um *framework* é uma estrutura de suporte definida em que um outro projeto de software pode ser organizado e desenvolvido. Um *framework* pode incluir programas de suporte, bibliotecas de código, linguagens de *script* e outros softwares para ajudar a desenvolver e juntar diferentes componentes de um projeto de software (ASTELS; MILLER; NOVAK, 2002, p. XXIV).

[6] Todos os envolvidos em um projeto de desenvolvimento de software. Usuários e clientes podem ser considerados *stakeholders* em um projeto de software.

a) compacto: apenas aspectos relevantes estão definidos;
b) completo: atinge todas as fases de ciclo de vida de desenvolvimento de software;
c) extensível: este processo pode ser utilizado na forma original em que foi disponibilizado, entretanto podem ser adicionadas outras atividades relacionadas conforme a necessidade de um determinado projeto ou instituição.

O conteúdo do OpenUP estabelece suas bases em quatro áreas de conteúdo que são a colaboração e comunicação, intenção, gerenciamento e solução. Todas as áreas de conteúdo possuem aspectos grandiosos da participação dos envolvidos no processo de desenvolvimento.

2.3.1 Princípios do OpenUP

Conforme Balduino (2007), os princípios capturam as intenções gerais por detrás de um processo e criam a base para interpretar papéis e produtos de trabalho. O OpenUP é dirigido por quatro princípios essenciais, os quais são:

a) colaboração para alinhar interesses e compartilhar compreensão: este princípio promove práticas que fomentam um ambiente de equipe saudável, possibilitando a colaboração e desenvolvendo uma compreensão compartilhada do projeto;
b) equilíbrio de prioridades concorrentes para maximizar valor ao *stakeholder*: este princípio promove práticas que permitem aos participantes do projeto e aos *stakeholders* desenvolverem uma solução que maximize os benefícios ao *stakeholder* e esteja de acordo com as restrições postas ao projeto;
c) focalização na arquitetura para minimizar riscos e organizar o desenvolvimento: este princípio promove práticas que permitem à equipe focalizar na arquitetura a fim de minimizar riscos e organizar o desenvolvimento;
d) evolução contínua para obter avaliações e melhoras: este princípio promove práticas que permitem à equipe receber avaliações dos *stakeholders* cedo e continuamente e demonstrar para eles valor incremental.

Cada princípio do OpenUP sustenta uma declaração no manifesto de agilidade[7], como

[7] É uma forma mais eficiente de construir software, desenvolvendo e ajudando outros a desenvolverem.

poder ser visto no Quadro 1.

PRINCÍPIO DO OPENUP	**DECLARAÇÃO DO MANIFESTO DE AGILIDADE**
Colaboração para alinhar interesses e compartilhar compreensão.	Indivíduos e interações sobre processos e ferramentas.
Equilíbrio de prioridades concorrentes para maximizar valor ao *stakeholder*.	Colaboração do cliente na negociação do contrato.
Focalização na arquitetura para minimizar riscos e organizar o desenvolvimento.	Trabalhar o software em cima de ampla documentação.
Evolução para continuamente obter avaliações e melhoras.	Responder a mudanças seguindo um plano.

Fonte: adaptado de Balduino (2007).

Quadro 1 – Mapeamento entre os princípios OpenUP e o manifesto de agilidade

2.3.2 Organização do OpenUP

O OpenUP é organizado em duas dimensões diferentes e correlatas, os quais são: conteúdo de método e conteúdo de processo. O conteúdo de método é onde os elementos de método, como papéis, tarefas, artefatos e orientação são definidos, independente de como eles são utilizados no ciclo de vida de um projeto. O conteúdo de processo é onde os elementos de método são aplicados em um sentido temporal. Muitos ciclos de vida diferentes para diferentes tipos de projeto podem ser criados do mesmo conjunto de elementos de método.

2.3.3 Áreas de conteúdo

O conteúdo do OpenUP aborda a organização do trabalho em nível pessoal, de equipe e de *stakeholder*.

Conforme Balduino (2007), em um nível pessoal, os membros de equipe em um projeto OpenUP levam o seu trabalho em micro incrementos, que normalmente representam o resultado de algumas horas a alguns dias de trabalho. A aplicação evolui um micro incremento por vez e o progresso é visto efetivamente todos os dias. Os membros da equipe compartilham o seu progresso diário nos micro incrementos, o que aumenta a visibilidade do trabalho, a confiança e o trabalho de equipe.

O OpenUP estrutura o ciclo de vida do projeto em quatro fases: concepção;

elaboração; construção e transição. O ciclo de vida do projeto aparelha os *stakeholders* com supervisão, transparência e mecanismos para controlar o financiamento do projeto, âmbito, exposição a riscos e outros aspectos do processo.

2.3.4 Processo

Conteúdo de método reutilizável é criado separadamente de sua aplicação nos processos. O conteúdo de método provê explicações passo a passo, descrevendo como metas de desenvolvimento específicas são alcançadas independentemente da colocação de elementos de método dentro de um ciclo de vida de desenvolvimento.

Os processos levam estes elementos de método e os relacionam em sucessões semi-ordenadas, que são personalizadas a tipos específicos de projetos. Os elementos de método são organizados em peças de processo reutilizáveis chamadas padrões de capacidade, providenciando uma abordagem de desenvolvimento consistente a necessidades comuns de projeto. Estes padrões são feitos através da organização de tarefas (do conteúdo de método) em atividades, agrupando-as numa seqüência que faça sentido para a área em particular onde tais padrões sejam aplicados.

Os padrões podem ser pequenos e focados em áreas particulares como gerenciamento de iteração, início de projeto, definição de arquitetura e assim por diante. Estes são considerados os blocos básicos para criar padrões maiores ou processos de entrega.

Conforme Balduino (2007), um exemplo de bloco básico no OpenUP é o padrão “desenvolva incremento de solução”, como ilustra a Figura 8.

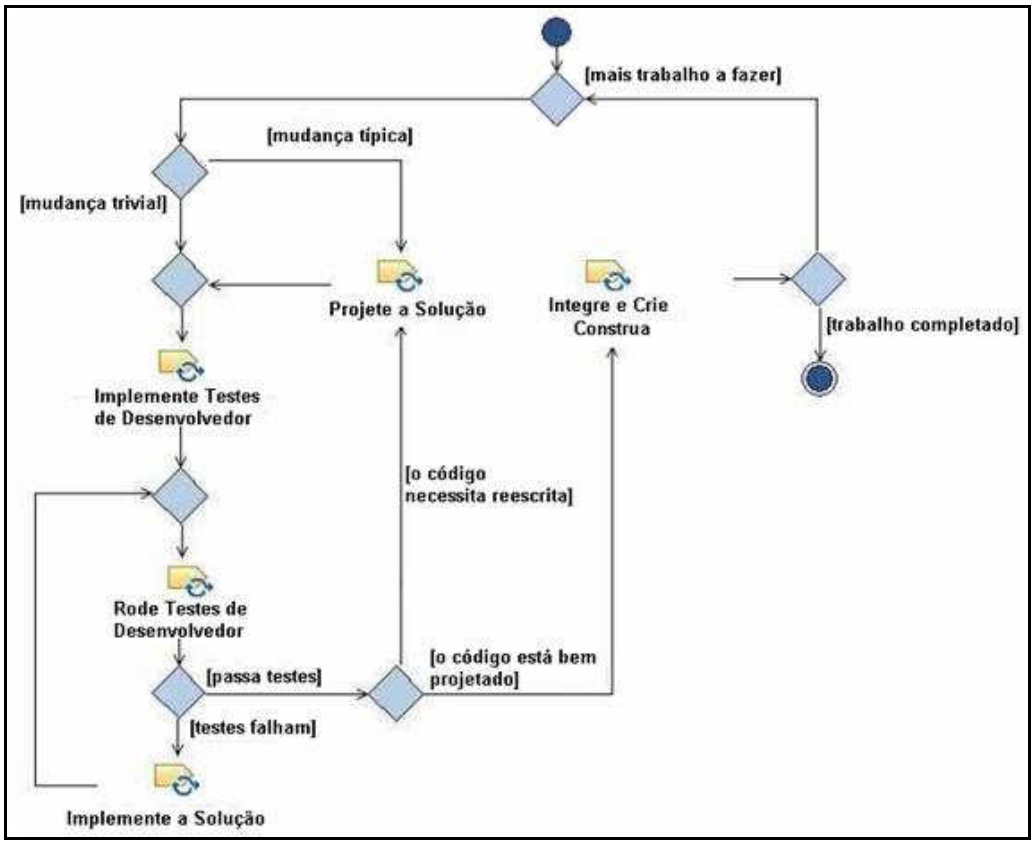
Esta atividade proporciona uma maneira de realizar planejamento e execução de trabalho baseado em metas. O trabalho é assumido por desenvolvedores e o progresso do trabalho é acompanhado baseado nas metas alcançadas utilizando o código fonte, projetado, testado pelo desenvolvedor e integrado.

O item de trabalho pode ser um caso de uso, um cenário, um requisito de suporte ou um pedido de mudança. Um contexto pode ser especificado quando se decide desenvolver um item de trabalho, especificando desta forma quão amplamente um item de trabalho deverá ser desenvolvido naquele incremento.

O desenvolvimento pode ser focado numa camada (interface de usuário, lógica de

negócio ou acesso à base de dados), ou num componente. Se um contexto é específico ou não, a responsabilidade do desenvolvedor é criar um *design* e uma implementação para aquele item de trabalho, também escrever e rodar testes de desenvolvedor contra a implementação para assegurar-se de que a implementação funcione conforme o projetado, tanto como unidade como na base de código.

O padrão “desenvolva incremento de solução” ocorre tantas vezes quanto houver itens de trabalho a serem desenvolvidos em uma dada iteração.



Fonte: adaptado de Balduino (2007).

Figura 8 – Desenvolva incremento de solução

Conforme mencionado, blocos básicos são utilizados para criar padrões maiores, padrões que, por exemplo, possam ser utilizados como *templates* para iterações, ou seja, padrões contendo todas as atividades necessárias para uma iteração em particular dentro de uma fase do projeto.

2.3.5 Um processo base

Conforme Balduino (2007), um processo OpenUP, embora completo em sua cobertura de objetivo e em seu contexto, também serve como um processo base sobre qual o conteúdo de processo adicional possa ser desenvolvido. Os *plug-ins* podem estender o OpenUP para incluir orientação para técnicas de larga escala (tais como desenvolvimento dirigido por modelo) e técnicas mais leves e ágeis (tais como técnicas de bases de dados ágeis). Os vendedores de ferramentas podem criar *plug-ins* que agregam procedimentos para tarefas dando instruções passo a passo de como a ferramenta possa ser utilizada dentro do contexto do processo.

Em adição a este conjunto de *plug-ins* que podem ser misturados e juntados para desenvolver um processo personalizado, e possivelmente mais importante para a meta de ter um processo que atinja as necessidades específicas de um projeto em particular, a ferramenta EPFC pode ser utilizada para criar conteúdo conforme a necessidade. Os *templates* específicos da organização podem ser integrados no conteúdo de processo e novos elementos de processo de *checklists* e de orientações para registrar novos papéis, tarefas e produtos de trabalho podem ser introduzidos.

2.4 ECLIPSE PROCESS FRAMEWORK COMPOSER

O EPFC, conforme Papo (2007, p. 31), é uma ferramenta que permite o desenvolvimento de processos e *framework* de processos podendo, ou não, utilizar o modelo OpenUP. O EPFC é um projeto desenvolvido pela EF.

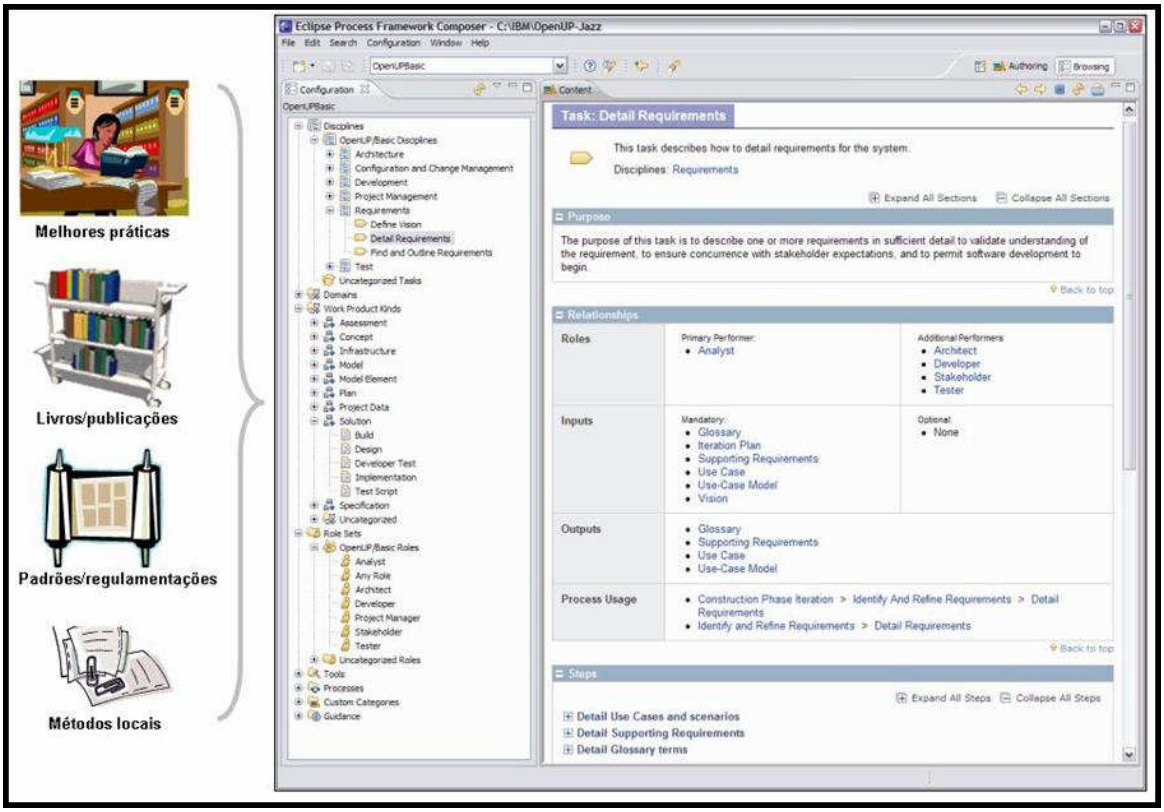
Papo (2007, p. 31) acrescenta que essa “ferramenta ajuda na autoria de métodos, autoria de processos, gestão de configuração de bibliotecas de métodos e configuração e publicação de processos.”

A Figura 9 ilustra como o conteúdo do método é representado no EPFC. Muitos métodos de desenvolvimento são descritos em publicações como livros, artigos, materiais de treinamento, padronizações e regulamentações. Estas fontes geralmente documentam métodos incluindo explicações em uma forma particular para atingir um objetivo de desenvolvimento

específico sob circunstâncias gerais. Alguns exemplos são: transformar os requisitos de um documento em um modelo de análise; definir um mecanismo arquitetural baseado em requisitos funcionais e não funcionais, e criar um plano de projeto para desenvolvimento em iterações.

O EPFC utiliza conteúdos conforme descrito anteriormente e os estrutura em um esquema específico de funções. Este esquema sustenta a organização de uma imensa quantidade de descrições para métodos de desenvolvimento e processos. Tal conteúdo e processos do método não precisam ser limitados à ES, mas também cobrir outros projetos e disciplinas criadas (ECLIPSE, 2007a).

A Figura 9 mostra como os elementos do conteúdo do método estão organizados em navegadores à esquerda. Estes navegadores proporcionam diferentes índices dos elementos disponíveis para acesso rápido. Na Figura 9 à direita, um exemplo de apresentação de tarefa. Esta apresentação define a tarefa em termos de passos que precisam ser executados para atingir seus objetivos. Pode-se observar que a tarefa tem relação com a execução de funções e com os produtos de trabalho, que servem como entrada e saída da mesma (ECLIPSE, 2007a).



Fonte: adaptado de Eclipse (2007a).

Figura 9 – Interface do EPFC

2.5 ENSINO DE ENGENHARIA DE SOFTWARE NA FURB

Em disciplinas de informática é necessário aliar teoria e prática para que os alunos consigam assimilar o conhecimento transferido. O ensino da ES utilizando trabalhos práticos é um fator de motivação para os alunos. Os trabalhos práticos em equipe têm a função de desenvolver características não técnicas do futuro profissional, mas que são extremamente essenciais para sua carreira, ou seja, sintetizar, escrever e apresentar oralmente resultados obtidos (SOARES, 2004).

As disciplinas de engenharia de software, requisitos de software e projetos de software são oferecidas semestralmente aos acadêmicos dos cursos das áreas de informática da FURB. Estas disciplinas possuem necessidades relevantes para um bom aproveitamento do acadêmico ao longo do seu aprendizado, tais como, a definição de um processo de software para o ensino, visando sua utilização em projetos de pequeno porte e a definição de um roteiro didático para auxiliar os acadêmicos no estudo da ES e realizar a integração entre os professores.

A ES, devido a sua abrangência de conhecimentos é normalmente abordada através de diversas disciplinas ao longo dos cursos de ciências da computação e sistemas de informação. A FURB possui em sua grade seis disciplinas associadas a ES, sendo três oferecidas ao curso de ciências da computação e três ao curso de sistemas de informação. Na seção 2.5.1 são apresentadas as disciplinas da área de ES da FURB que são oferecidas no novo currículo 2007/1 aos cursos de ciências da computação e sistemas de informação.

2.5.1 Disciplinas oferecidas no novo currículo 2007/1

São oferecidas ao curso de ciências da computação três disciplinas associadas a ES, as quais são: engenharia de software; processo de software I e processo de software II.

A disciplina de engenharia de software direciona-se ao estudo de processos de desenvolvimento de software, modelos e normas de qualidade de software, gerência de projetos, engenharia de requisitos, gerência de configuração e mudanças, verificação, validação e testes. Além disso, aborda-se a utilização de ferramentas de *Computer Aided*

Software Engineering[8] (CASE). A meta desta disciplina é apresentar métodos e técnicas empregados no desenvolvimento de software em uma visão que segue um sistema ordenado, incluindo o gerenciamento de projetos e qualidade do produto e do processo (FURB – CIÊNCIAS DA COMPUTAÇÃO, 2008a).

A disciplina de processo de software I está focada em processo de desenvolvimento de software, documentação e especificação de requisitos, modelo de análise e medição. O objetivo desta disciplina é conhecer as etapas de um processo de desenvolvimento de software e praticar atividades voltadas à etapa de análise de requisitos de software aplicados à área de computação (FURB – CIÊNCIAS DA COMPUTAÇÃO, 2008b).

Em processo de software II serão apresentados métodos e técnicas para projeto de software, construção de software, testes de software e documentação do usuário. O objetivo desta disciplina é praticar as atividades voltadas às etapas de projeto, implementação e testes de um software aplicado na área de computação (FURB – CIÊNCIAS DA COMPUTAÇÃO, 2008c).

Ao curso de sistemas de informação são oferecidas três disciplinas associadas a ES, as quais são: engenharia de software I; projeto de software I e projeto de software II.

A disciplina de engenharia de software I abrange os conteúdos de processos de desenvolvimento de software, gerência de projetos, engenharia de requisitos, gerência de configuração e mudança, verificação, validação e testes. O principal foco desta disciplina é conhecer métodos e técnicas aplicados ao desenvolvimento de software de uma maneira sistemática, inserindo gerenciamento de projetos e qualidade do produto e do processo (FURB – SISTEMAS DE INFORMAÇÃO, 2008a).

A disciplina de projeto de software I compreende os conteúdos de processo de desenvolvimento de software, documento de especificação de requisitos, modelo de análise, medição e padrões de projeto. O objetivo desta disciplina é abordar as etapas de um processo de desenvolvimento de software. Para promover a integração interdisciplinar, praticar atividades voltadas à etapa de análise de requisitos de aplicativos (FURB – SISTEMAS DE INFORMAÇÃO, 2008b).

Em projeto de software II serão apresentados temas com relação a projeto de software, construção de software, teste de software e documentação do usuário. O objetivo desta disciplina é requerer a integração interdisciplinar, praticar atividades voltadas às etapas de

[8] CASE significa engenharia de software auxiliada por computador.

projeto, especificação, implementação e teste de um aplicativo web (FURB – SISTEMAS DE INFORMAÇÃO, 2008c).

No Quadro 2 pode-se observar a relação entre os temas abordados nas ementas das disciplinas do currículo 2007/1 com as principais disciplinas do processo OpenUP. A aderência das disciplinas da área de ES da FURB com o OpenUP reforça sua utilização como processo base na publicação do FurbUP.

EMENTAS	OpenUp – Disciplinas	Grau de Atendimento	Observação
Engenharia de Software – BCC			
Processos de desenvolvimento de software.	Processo OpenUp	Alto	
Modelos e normas de qualidade de software.	Requisitos	Médio	Somente foca norma de produto de software, porém considerando o CMMI e MPS.Br vários processos são previstos no OpenUp – Requisitos, Gerenciamento de Projeto, Teste, Arquitetura, Desenvolvimento.
Gerência de projetos.	Gerenciamento de Projeto	Alto	
Engenharia de requisitos.	Requisitos	Alto	
Gerência de configuração e mudanças.	Gerência de configuração e mudanças.	Alto	
Verificação, validação e testes.	Teste	Alto	
Engenharia de Software I – SIS			
Processos de desenvolvimento de software.	Processo OpenUp	Alto	
Gerência de projetos.	Gerenciamento de Projeto	Alto	
Engenharia de requisitos.	Requisitos	Alto	
Gerência de configuração e mudanças.	Gerência de configuração e mudanças.	Alto	
Verificação, validação e testes.	Teste	Alto	
Processo de Software I – BCC			
Processo de desenvolvimento de software.	Processo OpenUp	Alto	
Documentação de especificação de requisitos.	Requisitos	Alto	
Modelo de análise.	Requisitos	Alto	
Medição.	Gerenciamento de Projeto	Baixo	
Projeto de Software I – SIS			
Processo de desenvolvimento de software.	Processo OpenUp	Alto	
Documento de especificação de requisitos.	Requisitos	Alto	
Modelo de análise.	Requisitos	Alto	
Medição.	Gerenciamento de Projeto	Baixo	
Padrões de projeto.	Arquitetura	Alto	
Processo de Software II – BCC			
Projeto de software.	Arquitetura	Alto	
Construção de software.	Desenvolvimento	Alto	
Teste de software.	Teste	Alto	
Documentação do usuário.	Requisitos	Baixo	Podem ser reusados os cenários de casos de uso, mas não foca manual do usuário e nem documentos de ajuda.
Projeto de Software II – SIS			
Projeto de software.	Arquitetura	Alto	
Construção de software.	Desenvolvimento	Alto	
Teste de software.	Teste	Alto	
Documentação do usuário.	Requisitos	Baixo	Podem ser reusados os cenários de casos de uso, mas não foca manual do usuário e nem documentos de ajuda.

Quadro 2 – Relação entre os temas das disciplinas do currículo 2007/1 com o OpenUP

Alguns conteúdos com baixo grau de atendimento serão amenizados pela inclusão de diretrizes e novas atividades no FurbUP.

2.6 TRABALHOS CORRELATOS

Alguns processos de desenvolvimento desempenham papel semelhante ao proposto no presente trabalho, cada qual com suas particularidades e objetivos específicos. Dentre eles cita-se: easYProcess (YP); ProcessID[9] (PID) e MetoDes (MD).

2.6.1 easYProcess

Garcia et al. (2004) desenvolveram um processo de software com o objetivo de utilização no ambiente acadêmico. O YP é um processo de desenvolvimento de software simplificado, apoiado em práticas de RUP, XP e *Agile Modeling*[10] (AM). Este trabalho mostra o YP como um processo para a academia que foi aplicada no departamento de sistemas e computação da Universidade Federal de Campina Grande (UFCG).

A criação deste processo surgiu das dificuldades encontradas em se adaptar vários processos já concebidos para uso acadêmico. Estas dificuldades foram observadas nas disciplinas de ES do curso de ciências da computação da UFCG.

O YP foi elaborado a partir de três fases, as quais são: estudo; concepção e implantação. Os estudos foram diretamente focados na observação da carência da utilização de processos de desenvolvimento em empresas, pelas características de processos já existentes e pelo que estava sendo abordado na literatura referente à ES.

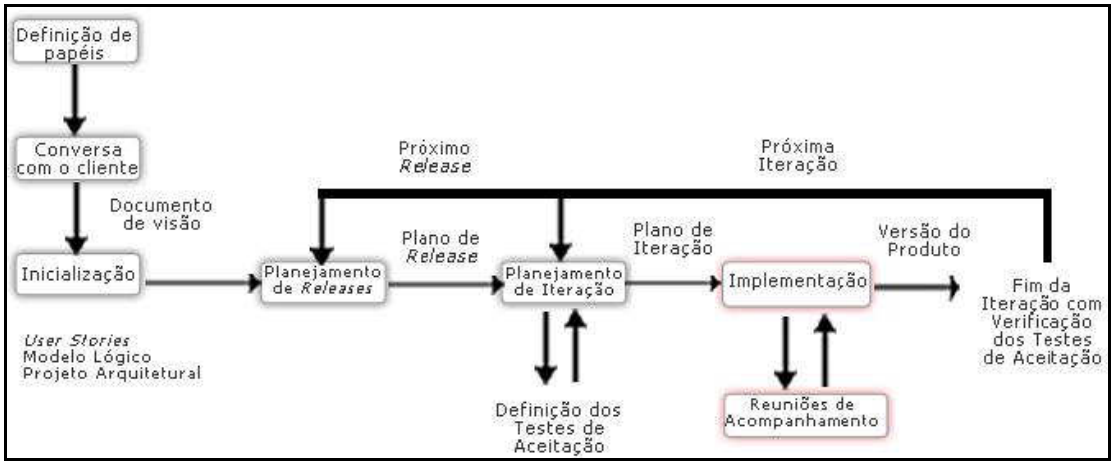
Com relação aos processos já existentes, como o RUP, XP e AM, foram analisados seus artefatos e atividades, com o objetivo de extrair os conteúdos fundamentais, importantes de serem adotados em projetos de pequeno e médio porte.

O fluxo básico do YP está ilustrado na Figura 10, elaborado na fase de concepção do processo.

---

[9] Nome criado para o processo de desenvolvimento integrando disciplinas de ES, sendo que este será utilizado apenas no contexto deste trabalho. Isto foi feito, pois o referido processo não possui um nome.

[10] AM é um processo baseado em princípios que definem critérios para os processos de desenvolvimento ágil de software. A AM ajuda a encontrar um meio termo, no qual se modela o suficiente para explorar e documentar um sistema de modo eficaz (AMBLER, 2003, p. 23).



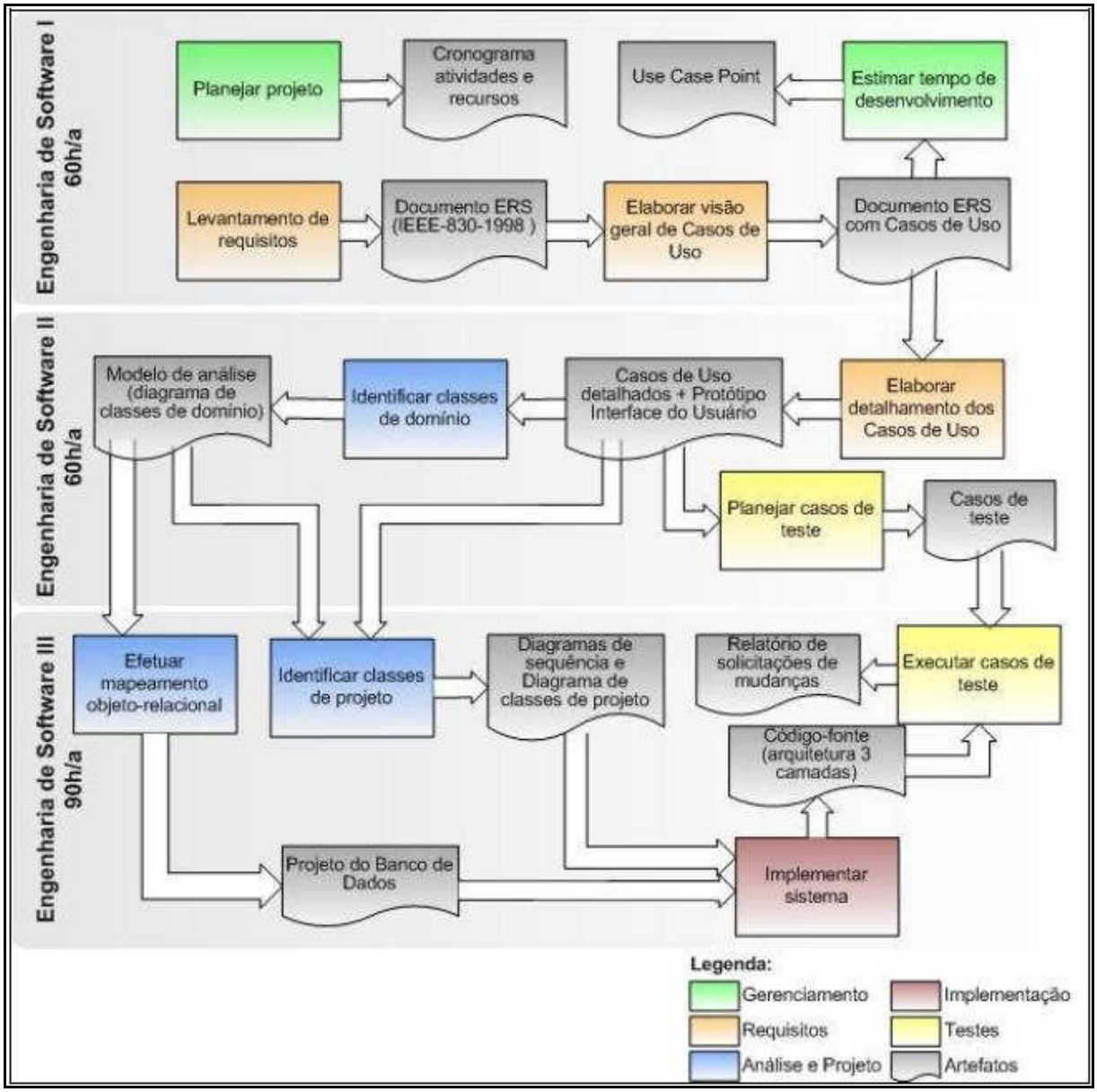
Fonte: Garcia et al (2004).

Figura 10 – Fluxo do YP

2.6.2 ProcessID

Alves e Benitti (2006) apresentaram um processo de desenvolvimento que tem o objetivo de integrar as disciplinas de ES em um curso de ciências da computação. O PID promove a participação do aluno com os conteúdos e práticas da ES desde a primeira disciplina. Como resultado da metodologia estudada é desenvolvido um projeto de software que reúne gerência de projeto, cálculo de medições, análise, projeto, implementação e testes. Com a prática deste processo de desenvolvimento os alunos aprendem cada vez mais a trabalhar em equipes e os professores podem trabalhar de forma integrada.

No PID o planejamento das disciplinas de ES acontece de forma integrada em três semestres subseqüentes, envolvendo duas professoras diferentes, em um único processo que proporciona aos alunos, além dos conceitos teóricos abordados, a prática na elaboração de um projeto de software assimilando as várias fases de um processo conciso, conforme ilustra a Figura 11.



Fonte: Alves e Benitti (2006).

Figura 11 – Fluxo do PID

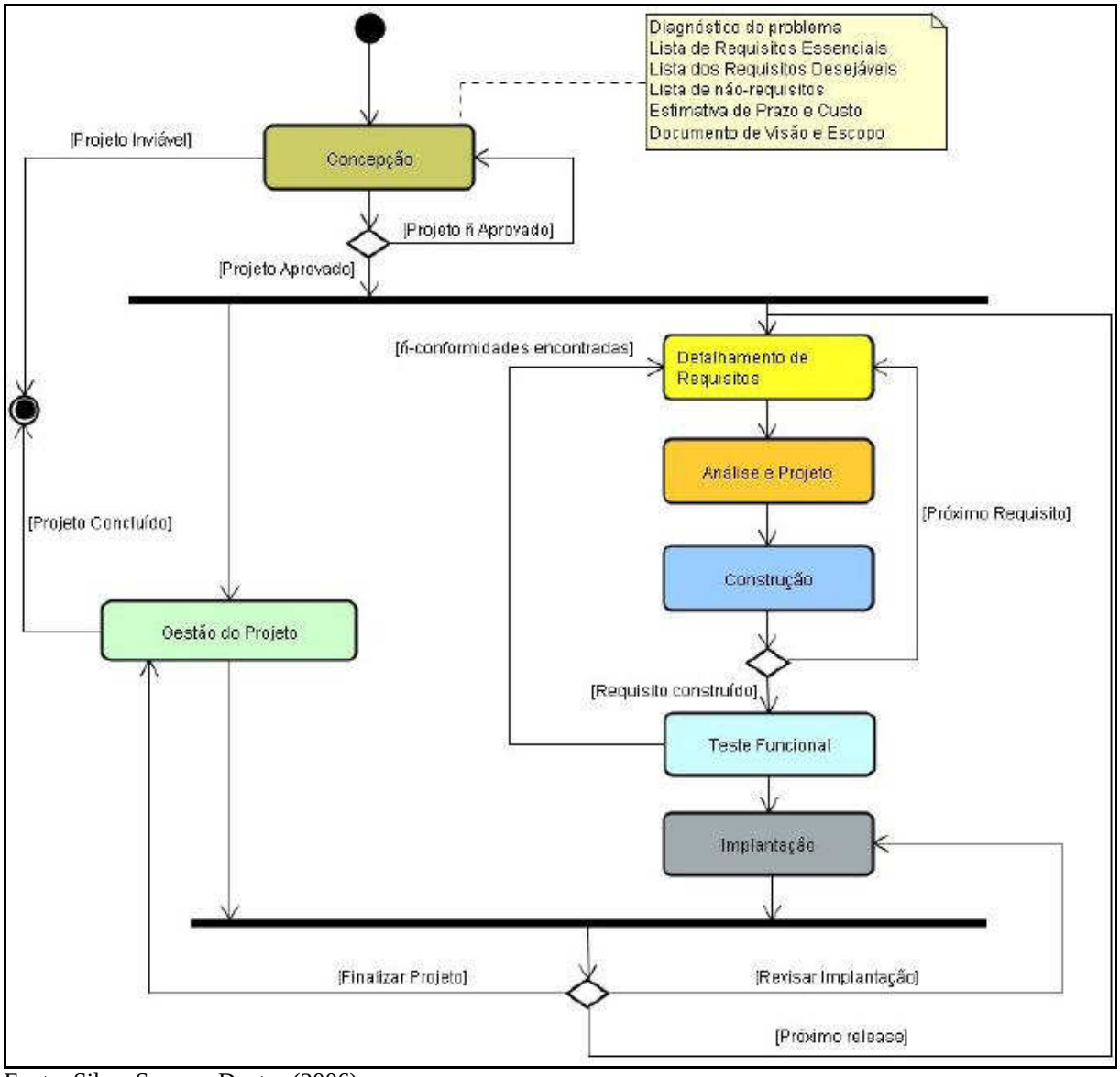
2.6.3 MetoDes

Silva, Souza e Dantas (2006) elaboraram uma metodologia de desenvolvimento de software, com o objetivo de utilização no ambiente acadêmico da Faculdade Cenecista de Brasília (FACEB). A MD aborda práticas do desenvolvimento iterativo e incremental de software, utilizando a UML como linguagem de modelagem de sistemas. O principal objetivo da MD é disponibilizar aos acadêmicos da FACEB uma metodologia de desenvolvimento de

software que facilite a compreensão das atividades envolvidas no desenvolvimento de projetos de software.

A MD foi baseada em processos comerciais para desenvolvimento de software, utilizados em alta escala no mercado, os quais são: RUP e XP. Assim, fundamentada nos conhecimentos da ES, a MD foi contextualizada levando em consideração as reais necessidades enfrentadas pelos acadêmicos do curso de sistemas de informação.

A Figura 12 representa uma visão geral de como se dá o processo de desenvolvimento da MD e apresenta a visão dos relacionamentos entre as disciplinas da metodologia.



Fonte: Silva, Souza e Dantas (2006).

Figura 12 – Relacionamento entre as disciplinas da MD

3 DESENVOLVIMENTO

Com base nos conceitos e materiais estudados descritos no capítulo anterior, o presente trabalho foi desenvolvido através das seguintes fases:

a) tradução das principais características do OpenUP para a língua portuguesa, utilizadas na publicação do processo FurbUP;
b) implementação e especificação do estudo de caso, representado através de um sistema de reserva de laboratórios;
c) elaboração do *template* para auxiliar os acadêmicos no desenvolvimento de aplicações utilizando o FurbUP.

3.1 TRADUÇÃO DAS PRINCIPAIS CARACTERÍSTICAS DO PROCESSO OPENUP

Com o intuito de utilizar totalmente o processo OpenUP na publicação do FurbUP, decidiu-se traduzir para a língua portuguesa os conteúdos essenciais em torno do processo, que são apresentados nas seções seguintes. Estes conteúdos foram extraídos do *plug-in* designado openup que acompanha a distribuição do ambiente EPFC. A versão do processo OpenUP utilizada foi a 1.0. Importante ressaltar que atualmente não existe uma tradução completa deste processo disponível em português. Existe apenas uma iniciativa de voluntários para realizar esta tradução no próprio site da EF, porém a tradução está em sua fase inicial e praticamente não evoluiu mais.

3.1.1 Introdução ao OpenUP

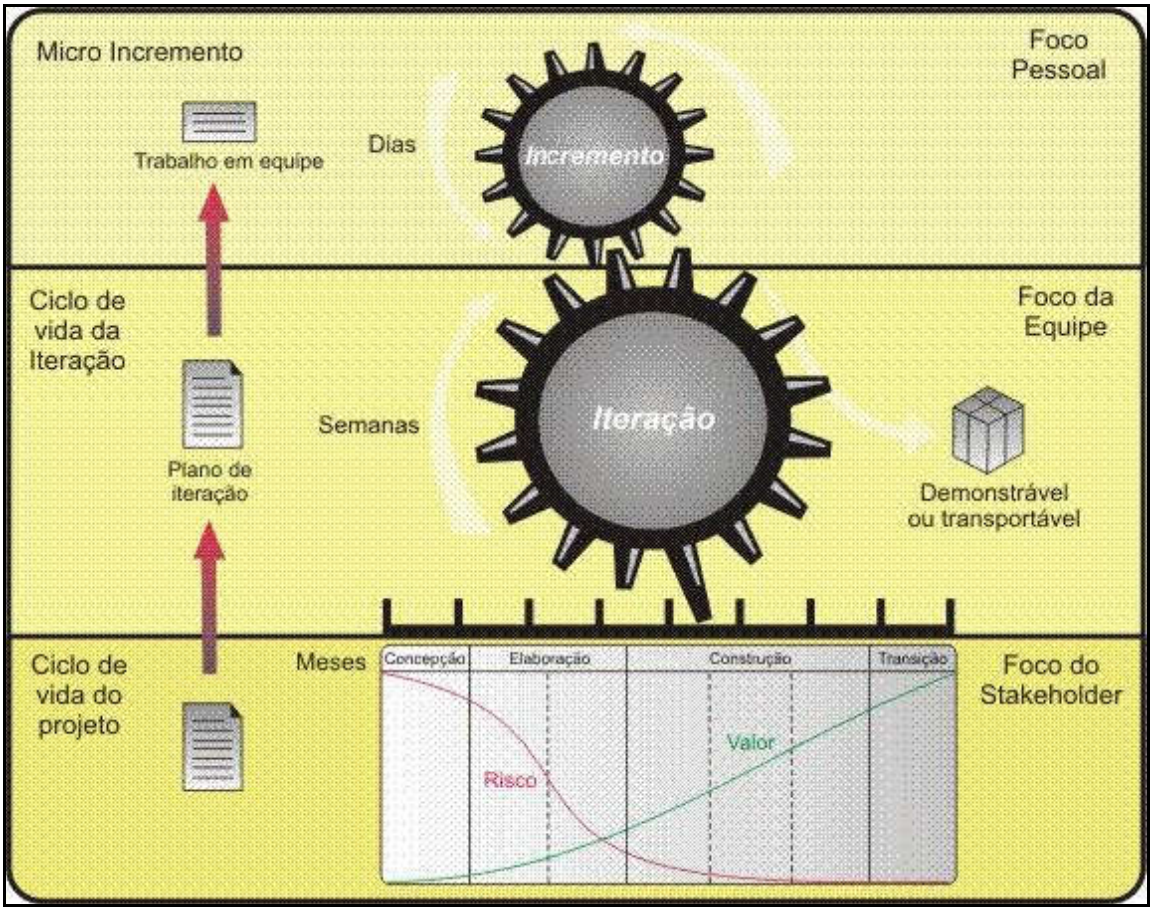
O OpenUP é um processo unificado que aplica abordagens iterativas e incrementais dentro de um ciclo de vida estruturado. O OpenUP abrange uma filosofia pragmática e ágil que foca na natureza colaborativa do desenvolvimento de software.

Em um projeto OpenUP o esforço pessoal é organizado em micro incrementos. Estes

representam pequenas unidades de trabalho que produzem um ritmo de projeto constante e mensurável, medido tipicamente em horas ou em poucos dias.

O OpenUP divide o projeto em iterações, ou seja, intervalos de tempo definidos, planejados e normalmente medidos em semanas. As iterações focam a equipe na entrega de valor incremental aos *stakeholders* de modo previsível. As equipes OpenUP se auto-organizam ao redor de como realizar os objetivos da iteração e se comprometem a entregar resultados, conforme o plano de iteração. Elas o fazem definindo e extraindo tarefas detalhadas de uma lista de itens de trabalho. O OpenUP aplica um ciclo de vida de iteração que estrutura como os micro incrementos são aplicados para entregar construções do sistema, estáveis e coesivas, que progride incrementalmente aos objetivos da iteração (ECLIPSE, 2007b).

Em adição a este conjunto de características, o OpenUP estrutura o ciclo de vida do projeto em quatro fases, que são: concepção; elaboração; construção e transição. O ciclo de vida do projeto mune os *stakeholders* e membros da equipe com visibilidade e pontos de decisões ao longo do projeto. Isto propicia supervisão efetiva, o que permite a equipe tomar decisões importantes em momentos apropriados (ECLIPSE, 2007b). O conteúdo do OpenUP trata a organização do trabalho em nível pessoal, de equipe e de *stakeholder,* conforme ilustra a Figura 13.



Fonte: adaptado de Eclipse (2007b).

Figura 13 – Organização do trabalho e foco no conteúdo no OpenUP

O OpenUP está baseado em quatro princípios essenciais: colaboração; equilíbrio; foco e evolução. Esses princípios capturam e definem as intenções gerais do OpenUP, os quais criam a base para interpretação dos papéis, dos produtos de trabalho e para a realização de tarefas.

Alguns conceitos fundamentais se reúnem em torno do OpenUP, para um maior entendimento dos propósitos gerais do processo, os quais são:

a) conceito da fase de concepção: a primeira das quatro fases no ciclo de vida de projeto é o entendimento do escopo, dos objetivos do projeto e da obtenção de informações suficientes para confirmar que o projeto deva ser realizado;
b) conceito da fase de elaboração: segunda das quatro fases do ciclo de vida do projeto, quando riscos arquiteturais significantes são resolvidos;
c) conceito da fase de construção: terceira das quatro fases no ciclo de vida do projeto, a construção tem foco no *design*, implementação e testes das funcionalidades para desenvolver um sistema completo;
d) conceito da fase de transição: quarta e final fase no ciclo de vida do projeto, a

transição assegura que o software esteja pronto para ser entregue aos usuários;

e) conceito de iteração: é um período de tempo dentro de um projeto em que se produz uma versão estável e executável do produto, junto com toda a documentação de apoio necessários para usar a liberação. É também conhecida como ciclo ou tempo definido;
f) conceito de caso de uso: descreve o que o sistema tem que fazer para fornecer valor aos *stakeholders*;
g) conceito de risco: é qualquer coisa, desconhecida ou incerta, que possa impedir o sucesso do projeto. Geralmente, um risco é qualificado pela probabilidade da ocorrência e pelo impacto que pode causar no projeto, caso ocorra;
h) conceito de arquitetura de software: representa a estrutura ou as estruturas do sistema, que consiste em componentes de software, propriedades externamente visíveis dos componentes e os relacionamentos entre eles.

3.1.2 OpenUP disciplinas

Uma disciplina é uma coleção de tarefas que estão relacionadas a uma área de interesse principal dentro do projeto como um todo.

Conforme Balduino (2007), uma tarefa é uma unidade de trabalho que um dos papéis possa ser solicitado a realizar. No OpenUP há tarefas que os papéis executam, tanto como executores primários (o responsável por executar a tarefa) ou como auxiliares (apoiando e providenciando informações utilizadas na execução da tarefa). A natureza colaborativa do OpenUP é manifestada em ter os executores primários trabalhando com uma gama de outros indivíduos quando estão executando uma tarefa.

Agrupar tarefas em disciplinas é principalmente uma ajuda para compreender o projeto de uma tradicional perspectiva em cascata. Embora seja mais comum realizar tarefas simultaneamente por várias disciplinas (por exemplo, certas tarefas de requisitos são realizadas em coordenação cerrada com tarefas de análise e de *design*), separar estas tarefas em disciplinas é simplesmente um modo eficaz de organizar conteúdo, o que facilita a compreensão (ECLIPSE, 2007b).

Outra razão para que várias tarefas sejam categorizadas todas pela mesma disciplina é

que elas representam uma parte na realização de uma meta maior, ou elas executam trabalhos que são todos relacionados entre si. Cada disciplina define modos padronizados de realizar o trabalho que ela categoriza. Tais modos padrões são expressos pelos assim denominados fluxos de trabalho de referência descritos com padrões de capacidade, que definem como as tarefas categorizadas pela disciplina interagem (da maneira mais genérica). Estes fluxos de trabalho de referência são utilizados freqüentemente para educar e ensinar profissionais.

Assim como outros fluxos de trabalho, um fluxo de trabalho de referência de uma disciplina é uma seqüência semi-ordenada de atividades, apresentada ou como uma estrutura analítica ou como um diagrama de atividades realizadas para alcançar um resultado em particular. A natureza semi-ordenada dos fluxos de trabalho de disciplinas enfatiza que os fluxos de trabalho da disciplina não podem apresentar as reais nuances de agendamento de trabalho real, pois eles não podem descrever a opcionalidade de atividades, ou a natureza repetitiva de projetos reais. Ainda assim eles têm valor como um meio para compreender o processo, quebrando-o em menores áreas de interesse.

O OpenUP agrega uma lista de disciplinas que auxiliam a organizar as tarefas, as quais são:

a) arquitetura: explica como criar uma arquitetura a partir de requisitos arquiteturalmente importantes;
b) desenvolvimento: explica como projetar e implementar uma solução técnica que esteja de acordo com a arquitetura e atenda os requisitos;
c) gerência de configuração e mudança: explica como controlar as mudanças nos artefatos, assegurando uma evolução sincronizada de todos os produtos de trabalho que compõem um sistema de software;
d) gerenciamento de projeto: explica como treinar, facilitar e dar suporte à equipe, ajudando-a a lidar com riscos e obstáculos ao elaborar um software;
e) requisitos: define as tarefas mínimas necessárias para eliciar, analisar, especificar, validar e gerenciar os requisitos para o sistema a ser desenvolvido;
f) teste: explica como proporcionar *feedback* sobre o sistema em amadurecimento ao projetar, implementar, rodar e avaliar os testes.

3.1.3 OpenUP produtos de trabalho

O OpenUP possui uma lista de artefatos, ou seja, produtos de trabalho que são organizados em domínios, os quais são: arquitetura; desenvolvimento; gerenciamento de projeto; requisitos e teste. Um produto de trabalho é um elemento de conteúdo que representa qualquer coisa usada, produzida, ou modificada por uma tarefa. Os papéis são responsáveis pela criação e atualização dos artefatos (ECLIPSE, 2007b).

Os artefatos do OpenUP são considerados essenciais para capturar informações relacionadas ao produto e ao projeto. Não há obrigação da captura de informações em artefatos formais. As informações poderiam ser capturadas informalmente em quadro branco (para *design* e arquitetura, por exemplo), anotações de reuniões, etc. Os *templates*, entretanto proporcionam uma maneira pronta e padronizada para capturar informações. Os projetos podem utilizar os artefatos OpenUP ou substituí-los pelos seus próprios.

O domínio arquitetura agrega apenas um produto de trabalho, que é o caderno de arquitetura (anexo A). Este artefato descreve o contexto para o desenvolvimento de software, contendo as decisões, a análise racional, as explicações e as implicações em formar a arquitetura do sistema. Seu propósito é descrever o contexto e a perspectiva do sistema para garantir a integridade e compreensão.

O domínio desenvolvimento agrega quatro produtos de trabalho, os quais são:

a) construção: este artefato é uma versão operacional de um sistema ou de uma parte de um sistema que demonstre um subconjunto das capacidades a serem fornecidas no produto final. Seu propósito é entregar valor incremental ao usuário e cliente, bem como fornecer um artefato testável para verificação;
b) *design*: este artefato descreve a realização da funcionalidade requerida do sistema em termos de componentes e serve como uma abstração do código fonte. Seu propósito é descrever os elementos do sistema de forma que possam ser examinados e compreendidos de uma maneira não possível pela leitura do código fonte;
c) implementação: este artefato apresenta os arquivos de código do software, arquivos de dados e arquivos de suporte tais como os arquivos de ajuda *on-line* que representam as partes básicas de um sistema que possa ser construído. Seu propósito é representar as partes físicas que compõem o sistema a ser construído,

organizado de forma que seja compreensível e gerenciável;

d) teste de desenvolvedor: este artefato apresenta instruções que validam se os componentes individuais de software estão funcionando de acordo com o que foi especificado. Seu propósito é avaliar se um componente de software funciona de acordo com o que foi especificado.

O domínio gerenciamento de projeto agrega quatro produtos de trabalho, os quais são:

a) lista de itens de trabalho: este artefato contém uma lista de todo o trabalho agendado para ser feito dentro do projeto, bem como o trabalho proposto que pode afetar o produto de software (anexo B). Cada item de trabalho pode conter referências a informações que sejam relevantes para realizar o trabalho descrito no item de trabalho. Seu propósito é coletar todos os pedidos de trabalho que irão potencialmente ser executados no projeto, de forma que o trabalho possa ser priorizado, o esforço ser estimado e o progresso acompanhado;

b) lista de riscos: este artefato é uma lista de riscos conhecidos e expostos ao projeto, classificados em ordem de importância e associados com ações específicas de atenuação ou contingência (anexo C). Seu propósito é capturar riscos ao sucesso do projeto que foram percebidos;

c) plano de iteração: este artefato apresenta um plano detalhado descrevendo os objetivos, as atribuições de trabalho e critérios de avaliação para a iteração (anexo D). Seu propósito é comunicar os objetivos, a atribuição das tarefas e os critérios de avaliação para uma dada iteração;

d) plano de projeto: este artefato registra todas as informações necessárias para gerenciar o projeto (anexo E). Sua principal parte consiste em um plano genérico, contendo as fases e os marcos do projeto. Seu propósito é fornecer um documento central onde qualquer membro da equipe de projeto possa encontrar as informações sobre como o projeto será gerenciado.

O domínio requisitos agrega quatro produtos de trabalho, os quais são:

a) caso de uso: este artefato captura a seqüência das ações executadas por um sistema que tenham um resultado de valor observável para àqueles que interagem com o sistema (anexo F). Seu propósito é capturar o comportamento necessário ao sistema na perspectiva do usuário final, para conseguir um ou mais objetivos;

b) modelo de caso de uso: este artefato captura as ações executadas por um sistema através de modelos (diagramas de casos de uso) que descrevem seus

comportamentos (cenários). Também ajuda a orientar as várias tarefas do desenvolvimento de software no ciclo de vida;

c) especificação de requisitos suplementares: este artefato captura todos os requisitos do sistema não capturados nos cenários ou nos casos de uso, incluindo requisitos de atributos de qualidade e requisitos funcionais globais (anexo G);

d) glossário: este artefato define os termos importante usados no projeto. Estes termos são a base para a colaboração eficaz com os *stakeholders* e outros membros da equipe. Seu propósito é fornecer um vocabulário comum acordado por todos os *stakeholders*. Pode ajudar pessoas de diferentes grupos funcionais a alcançar uma compreensão mútua do sistema. O objetivo não é registrar todos os termos possíveis, mas somente aqueles que não estão claros, são ambíguos ou necessitem de elaboração;

e) visão: este artefato contém a definição da visão dos *stakeholders* a respeito do produto a ser desenvolvido, especificada em termos das principais características e necessidades dos *stakeholders* (anexo H). Seu propósito é fornecer uma base de alto nível, para os requisitos técnicos mais detalhados que são visíveis para os *stakeholders*. Captura a essência do sistema descrevendo requisitos de alto nível e restrições de *design* que dão ao leitor uma visão geral do sistema de uma perspectiva comportamental dos requisitos. Serve como entrada para o processo de aprovação do projeto.

O domínio teste agrega três produtos de trabalho, os quais são:

a) caso de teste: este artefato é a especificação de um conjunto de entradas de teste, condições de execução e resultados previstos, identificados com a finalidade de fazer a avaliação de algum aspecto particular de um cenário (anexo I). Seu propósito é fornecer uma maneira para capturar entradas de teste, condições e resultados esperados para um sistema. Identificar sistematicamente aspectos do software a serem testados e especificar se um resultado esperado foi alcançado baseado na verificação de um requisito do sistema, grupo de requisitos ou cenário;

b) registro de teste: este artefato coleta a saída bruta capturada durante uma execução única de um ou mais testes para uma única execução de um ciclo de testes. Seu propósito é verificar que um conjunto de testes tenha sido executado e fornecer informações relacionadas ao sucesso desses testes;

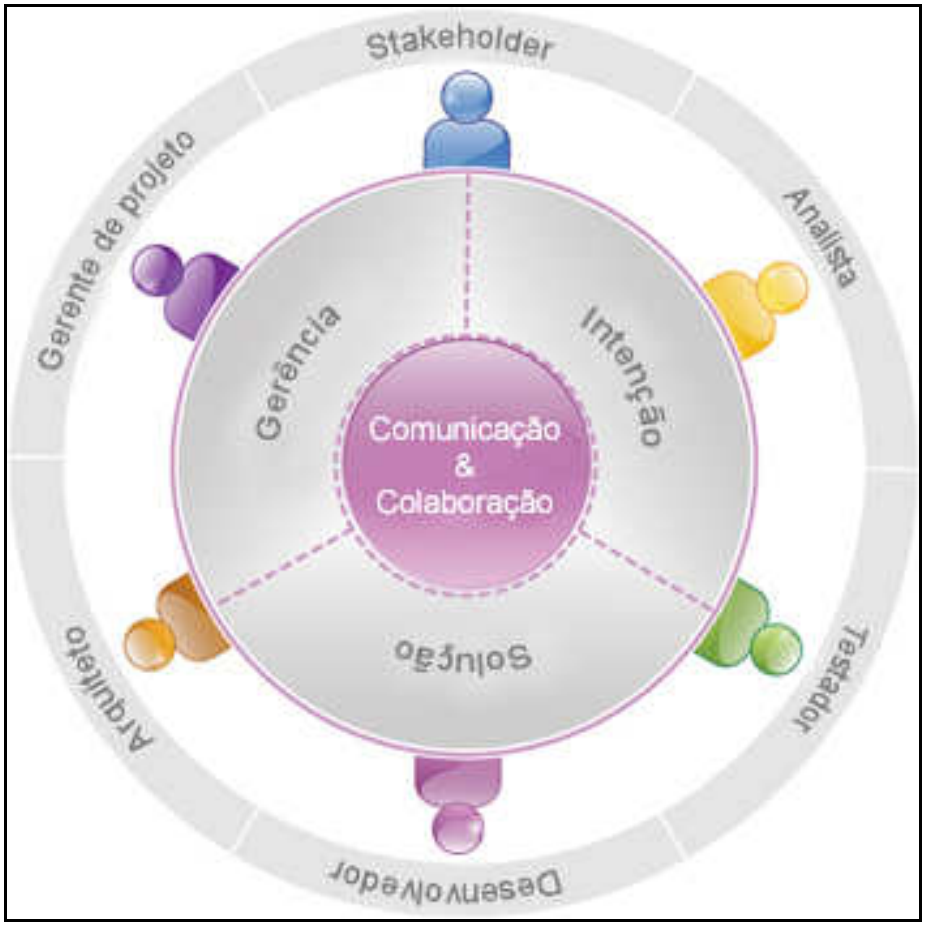
c) *script* de teste: este artefato contém instruções passo a passo para realizar um teste,

permitindo sua execução (anexo J). Pode ter a forma de instruções textualmente documentadas que são executadas manualmente ou instruções interpretáveis pelo computador que permitam a execução automatizada de testes.

3.1.4 OpenUP papéis

O Software é criado por pessoas com interesses e habilidades diferentes. Um ambiente de equipe saudável permite uma colaboração efetiva e requer uma cultura envolvida com criatividade e mudança positiva. Papéis são a face humana do processo de desenvolvimento de software. Além disso, são necessários novos pontos de vista para os papéis de projeto de software tradicionais, para alavancar a colaboração em vez de aumentar os canais de comunicação.

Papéis não representam responsabilidades individuais sobre tarefas ou resultados entregáveis, mas em vez disso são rótulos que as pessoas podem usar ao trabalharem juntas. Cada papel não é limitado a descrever o executor primário de alguma tarefa, em vez disso, os papéis incluem uma perspectiva de colaboração ao conseguir executores adicionais para cada tarefa. Representar um ou mais destes papéis pode ajudar as equipes a expressarem pontos de vista diferentes ao criar uma solução. Esta perspectiva de papéis dá força a uma nova geração de processos de desenvolvimento de software, mais focados na interação de pessoas. Ninguém faz um grande software sozinho, mas uma equipe trabalhando unida pode fazer coisas extraordinárias (ECLIPSE, 2007b). Os principais papéis no OpenUP e suas interações através das áreas de conteúdo, pode ser observada na Figura 14.



Fonte: adaptado de Eclipse (2007b).

Figura 14 – Principais papéis no OpenUP e suas interações

O OpenUP define um conjunto de sete papéis principais que são alocados para realização de tarefas no decorrer do processo de desenvolvimento, os quais são: analista; arquiteto; desenvolvedor; gerente de projeto; qualquer papel; *stakeholder* e testador.

O analista representa os interesses do cliente e dos usuários finais recolhendo informações dos *stakeholders* para entender o problema a ser resolvido, capturando os requisitos e definindo suas prioridades. Um analista necessita dos seguintes conhecimentos e habilidades:

a) perícia na identificação e entendimento de problemas e oportunidades;
b) habilidade de articular as necessidades que estão associadas com os principais problemas a serem resolvidos;
c) habilidade de colaborar eficazmente com toda a equipe em sessões colaborativas de trabalho, *workshops*, sessões *Joint Application Design* (JAD) e outras técnicas;
d) boa capacidade de comunicação verbal e escrita;
e) conhecimento dos domínios de negócio e de tecnologia ou habilidade de absorver e compreender rapidamente tal informação.

As principais tarefas alocadas para este papel são: definir a visão; detalhar requisitos e encontrar e esboçar requisitos. Com fundamento nessas tarefas o analista é responsável pelo desenvolvimento dos seguintes artefatos: modelo de caso de uso; especificação de requisitos suplementares; glossário e visão. As tarefas e artefatos relacionados ao analista podem ser observados na Figura 15.

Fonte: adaptado de Eclipse (2007b).

Figura 15 – Tarefas e artefatos relacionados ao analista

O arquiteto é responsável por definir a arquitetura do software, incluindo a tomada das principais decisões técnicas que controlam todo o *design* e a implementação do projeto. Os arquitetos devem ser pessoas sensatas com maturidade, visão e uma profunda experiência que permita coletar questões rapidamente, fazendo julgamentos educados e críticos no caso da ausência de informações completas. O arquiteto deve possuir especificamente a seguinte combinação de qualificações:

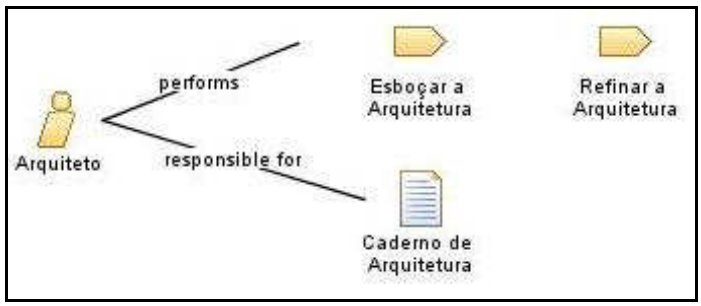
a) experiência nos domínios de problemas e de engenharia de software, com evidência de uma compreensão completa dos requisitos para resolver o problema e a participação ativa no desenvolvimento do software. Se houver uma equipe, esta experiência pode ser representada por membros diferentes da equipe, mas pelo menos uma pessoa deve estar capacitada a fornecer a visão total do projeto;
b) habilidade de liderança para motivar e manter constante o esforço técnico das várias equipes e tomar decisões críticas sob pressão, também fazer essas decisões valerem. Para ser eficaz, este papel deve ter autoridade para tomar decisões técnicas;
c) excelente habilidade de comunicação para ganhar a confiança, persuadir, motivar e ser mentor. Este papel não pode liderar por imposição, mas somente pelo consentimento de toda a equipe de projeto. Para ser eficaz, esta pessoa deve ganhar o respeito dos membros da equipe, do gerente de projeto, do cliente e da comunidade de usuários, bem como da equipe de gerenciamento;
d) orientação proativa e orientada a metas com um foco persistente em resultados. Esta pessoa é à força de condução por trás do projeto, não é um visionário ou um

> sonhador. A carreira de um arquiteto bem sucedido é uma longa série de decisões feitas sob incerteza e pressão. Somente aqueles que têm foco em fazer o que é necessário ser feito serão bem sucedidos.

De um ponto de vista pericial, este papel necessita também mostrar habilidades de *design* e de implementação. Entretanto, pela perspectiva de *design*, o arquiteto eficaz demonstra tipicamente os seguintes traços:

a) tende a ser um generalista, ao invés de um especialista, que conhece muitas tecnologias em alto nível ao invés de poucas tecnologias em nível detalhado;
b) toma as maiores decisões técnicas, demonstrando desse modo grande conhecimento e experiência, bem como habilidades de comunicação e liderança.

As principais tarefas alocadas para este papel são: esboçar a arquitetura e refinar a arquitetura. Com base nessas tarefas o arquiteto é responsável pelo desenvolvimento do seguinte artefato: caderno de arquitetura. As tarefas e artefatos relacionados ao arquiteto podem ser observados na Figura 16.



Fonte: adaptado de Eclipse (2007b).

Figura 16 – Tarefas e artefatos relacionados ao arquiteto

O desenvolvedor é responsável por desenvolver uma parte do sistema, incluindo a construção de seu *design* de forma que ele atenda a arquitetura e possivelmente a prototipagem da interface de usuário, e então implementar, executar o teste de unidade e integrar os componentes que são parte da solução.

As principais tarefas alocadas para este papel são: executar testes de desenvolvedor; implementar a solução; implementar testes de desenvolvedor; integrar e criar construção e projetar a solução. Também será responsável pela criação de manutenção dos seguintes artefatos: construção; *design*; implementação e teste de desenvolvedor. As tarefas e artefatos relacionados ao desenvolvedor podem ser observados na Figura 17.

Fonte: adaptado de Eclipse (2007b).

Figura 17 – Tarefas e artefatos relacionados ao desenvolvedor

O gerente de projeto conduz o planejamento do projeto, coordena as interações com os *stakeholders* e mantêm a equipe de projeto focada em alcançar os objetivos do projeto. Um gerente de projeto necessita das seguintes habilidades:

a) capacidades de liderança e de construção de equipe;
b) experiência completa no ciclo de vida de desenvolvimento de software para ensinar, guiar e auxiliar outros membros da equipe;
c) proficiente em solução de conflitos e em técnicas de solução de problemas;
d) ser bom em apresentação, facilitação, comunicação e negociação.

As principais tarefas alocadas para este papel são: avaliar resultados; gerenciar a iteração; planejar a iteração e planejar o projeto. Com fundamento nessas tarefas o gerente de projeto é responsável pelo desenvolvimento dos seguintes artefatos: lista de itens de trabalho; lista de riscos; plano de iteração e plano de projeto. As tarefas e artefatos relacionados ao gerente de projeto podem ser observados na Figura 18.

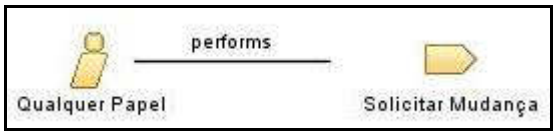
Fonte: adaptado de Eclipse (2007b).

Figura 18 – Tarefas e artefatos relacionados ao gerente de projeto

O papel denominado qualquer papel é definido como sendo qualquer um numa equipe que pode exercer e executar tarefas diversas, as quais são:

a) acessar artefatos no sistema de controle de configuração para desenvolvimento e manutenção;
b) submeter requisições de mudanças para o projeto;
c) participar das avaliações e revisões;
d) ser voluntário para trabalhar em uma determinada iteração.

A tarefa relacionada a qualquer papel pode ser observada na Figura 19.



Fonte: adaptado de Eclipse (2007b).

Figura 19 – Tarefa relacionada a qualquer papel

O *stakeholder* representa os grupos de interesse cujas necessidades devem ser atendidas pelo projeto. É um papel que deve ser executado por qualquer pessoa que é ou será afetada materialmente pelo resultado do projeto.

O testador é responsável pelas principais atividades do esforço de teste. Estas atividades incluem identificar, definir, implementar e conduzir os testes necessários, bem como registrar os resultados dos testes e analisar os resultados. Um testador necessita das seguintes habilidades:

a) conhecimento de abordagens e técnicas de teste;
b) habilidades de diagnóstico e solução de problemas;
c) conhecimento do sistema ou aplicação a ser testada;
d) conhecimento da arquitetura do sistema e de rede.

Caso testes automatizados sejam utilizados, o testador deverá possuir as seguintes qualificações adicionais:

a) treinamento no uso apropriado de ferramentas da automatização de teste;
b) experiência no uso de ferramentas da automatização de teste;
c) habilidades de programação;
d) habilidades de depuração e diagnóstico.

As principais tarefas alocadas para este papel são: criar casos de teste; executar os testes e implementar *scripts* de teste. Com base nessas tarefas o testador é responsável pelo desenvolvimento dos seguintes artefatos: caso de teste; registro de teste e *script* de teste. As tarefas e artefatos relacionados ao testador podem ser observados na Figura 20.

Fonte: adaptado de Eclipse (2007b).

Figura 20 – Tarefas e artefatos relacionados ao testador

3.1.5 OpenUP atividades

As atividades determinam o fluxo de tarefas e utilização e manutenção de artefatos por parte destas tarefas, devendo ser executadas pelas iterações em suas respectivas fases no ciclo de vida do desenvolvimento. O OpenUP, utiliza oito atividades distribuídas por sub-processos de gerência, intenção e solução, as quais são:

a) iniciar o projeto: faz parte do sub-processo de gerência. É realizada no começo da primeira iteração, quando o projeto inicia. O objetivo desta atividade é estabelecer a visão do projeto e o plano de projeto, em um elevado nível de granularidade;
b) planejar e gerenciar a iteração: faz parte do sub-processo de gerência. É realizada ao longo do ciclo de vida do projeto. O objetivo desta atividade é identificar riscos e questões cedo o suficiente para que possam ser mitigados, estabelecer as metas para a iteração e auxiliar a equipe de desenvolvimento a alcançar estas metas;
c) identificar e refinar requisitos: faz parte do sub-processo de intenção. Descreve as tarefas que são realizadas para juntar, especificar, analisar e validar um subconjunto de requisitos do sistema antes da implementação e da verificação. Isto não implica que todos os requisitos sejam detalhados antes de começar a implementação;
d) tarefas contínuas: faz parte do sub-processo de intenção. Esta atividade inclui uma única tarefa, solicitar mudança. Esta tarefa pode ocorrer a qualquer momento durante o ciclo de vida, em resposta a um defeito observado, a uma melhoria desejada ou a uma solicitação de mudança;
e) concordar na abordagem técnica: faz parte do sub-processo de solução. O objetivo desta atividade é definir uma abordagem técnica para o sistema que suporte os requisitos do projeto, em meio às restrições postas no sistema e na equipe de desenvolvimento;
f) desenvolver a arquitetura: faz parte do sub-processo de solução. Esta atividade refina a inicial arquitetura de alto nível em software operacional. O objetivo é produzir software estável que aborde adequadamente os riscos técnicos no escopo;
g) desenvolver incremento de solução: faz parte do sub-processo de solução. O objetivo desta atividade é projetar, implementar, testar e integrar a solução para um requisito dentro de um dado contexto. Um contexto pode ser especificado quando

um requisito for escolhido para ser desenvolvido, deste modo especificando quão amplamente um requisito deva ser desenvolvido em uma iteração;

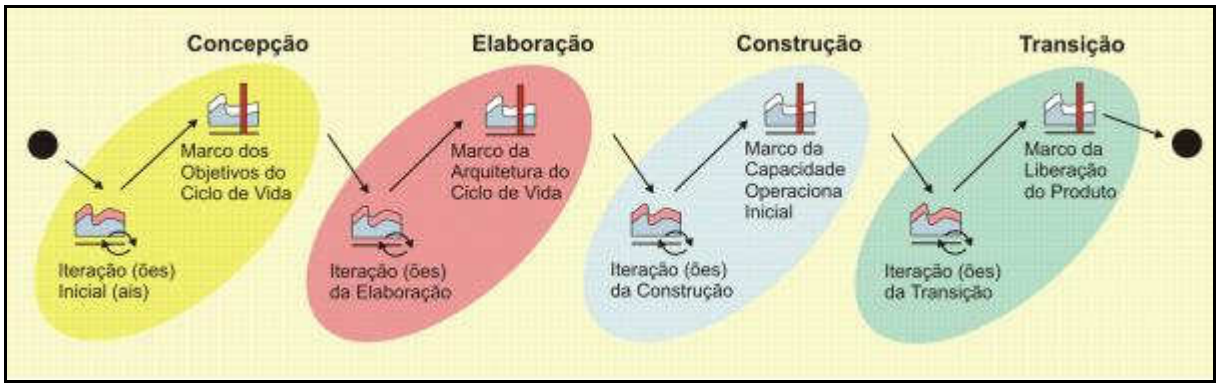
h) testar a solução: faz parte do sub-processo de solução. Esta atividade é repetida ao longo do ciclo de vida do projeto. O objetivo principal é validar que a corrente construção do sistema satisfaça os requisitos alocados para a mesma.

3.1.6 OpenUP ciclo de vida

O OpenUP é um processo iterativo com as iterações distribuídas por quatro fases, que são: concepção; elaboração; construção e transição.

Cada fase pode ter tantas iterações quantas forem necessárias, dependendo do grau de inovação do domínio do negócio, da tecnologia sendo utilizada, da complexidade arquitetural e do tamanho do projeto.

Quando se seqüencia os padrões de *templates* da iteração (ocorrendo tantas vezes quanto necessário), tem-se um processo de entrega, ou seja, uma abordagem completa e integrada para realizar um tipo de projeto especificado, conforme ilustra a Figura 21. Um processo de entrega descreve um ciclo de vida de projeto completo e é utilizado como uma referência para rodar projetos similares (BALDUINO, 2007).



Fonte: adaptado de Balduino (2007).

Figura 21 – Processo de entrega do OpenUP

Todas as fases do processo de desenvolvimento definem um conjunto de objetivos que devem ser alcançados por suas iterações, assim como a combinação com as atividades (seção 3.1.5) realizadas para atingir esses objetivos.

As iterações podem ter duração variável, dependendo das características do projeto. Iterações com um mês de duração são normalmente recomendadas, pois este período fornece:

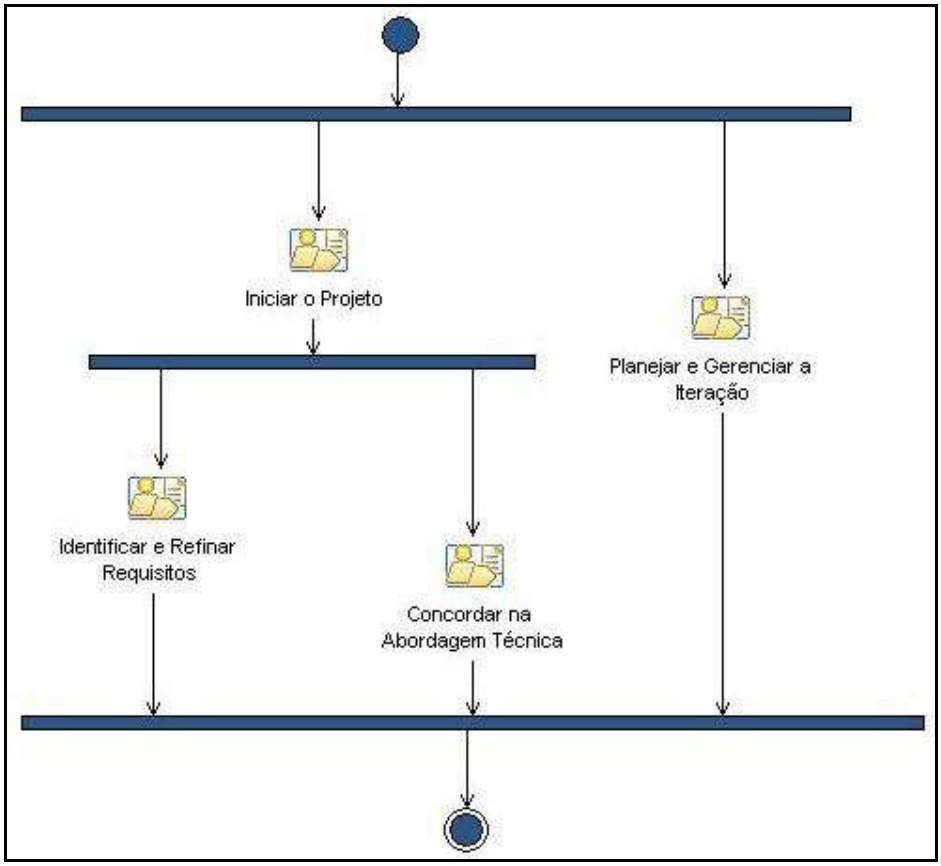
a) uma quantidade razoável de tempo para que os projetos possam entregar significativos incrementos de funcionalidade;
b) rápido e freqüente *feedback* dos clientes;
c) oportuna gestão de riscos e questões no decurso do projeto.

A fase de concepção é a primeira das quatro fases do ciclo de vida do processo. O propósito desta fase é conseguir entendimento simultâneo entre todos os *stakeholders* dos objetivos de ciclo de vida para o projeto.

Há quatro objetivos na fase de concepção que tornam claro o escopo, os objetivos do projeto e a viabilidade da solução pretendida, os quais são:

a) entender o que se quer construir. Determinar a visão, o escopo do sistema e seus limites;
b) identificar as funcionalidades chave do sistema. Decidir quais requisitos são mais críticos;
c) determinar pelo menos uma solução possível. Identificar pelo menos uma arquitetura candidata e sua aplicação prática;
d) entender o custo, cronograma, e os riscos associados ao projeto.

Para atingir os objetivos e resultados esperados durante a fase de concepção, uma série de atividades devem ser seguidas durante as iterações, as quais são: iniciar o projeto; planejar e gerenciar a iteração; identificar e refinar requisitos e concordar na abordagem técnica. O fluxo de atividades da fase de concepção pode ser observado na Figura 22.



Fonte: adaptado de Eclipse (2007b).

Figura 22 – Fluxo de atividades da fase de concepção

No Quadro 3 pode ser observado um resumo dos objetivos da fase de concepção e quais atividades endereçam cada objetivo.

Objetivos da fase	**Atividades que endereçam os objetivos**
- Compreender o que desenvolver. - Identificar funcionalidades chave do sistema. - Determinar ao menos uma solução possível. - Compreender o custo, o cronograma e os riscos associados ao projeto.	- Iniciar o projeto. - Planejar e gerenciar a iteração. - Identificar e refinar os requisitos. - Concordar na abordagem técnica.

Fonte: adaptado de Eclipse (2007b).

Quadro 3 – Objetivos e atividades da fase de concepção

O fim da fase de concepção é o primeiro grande marco do processo, o marco dos objetivos do ciclo de vida.

A fase de elaboração é a segunda das quatro fases do ciclo de vida do processo. O propósito desta fase é estabelecer uma linha de base da arquitetura do sistema e prover uma base estável para o volume de esforço de desenvolvimento na próxima fase (ECLIPSE, 2007b).

Há três objetivos na fase de elaboração que ajudam a resolver os riscos associados com requisitos, arquitetura, custos e cronograma, os quais são:

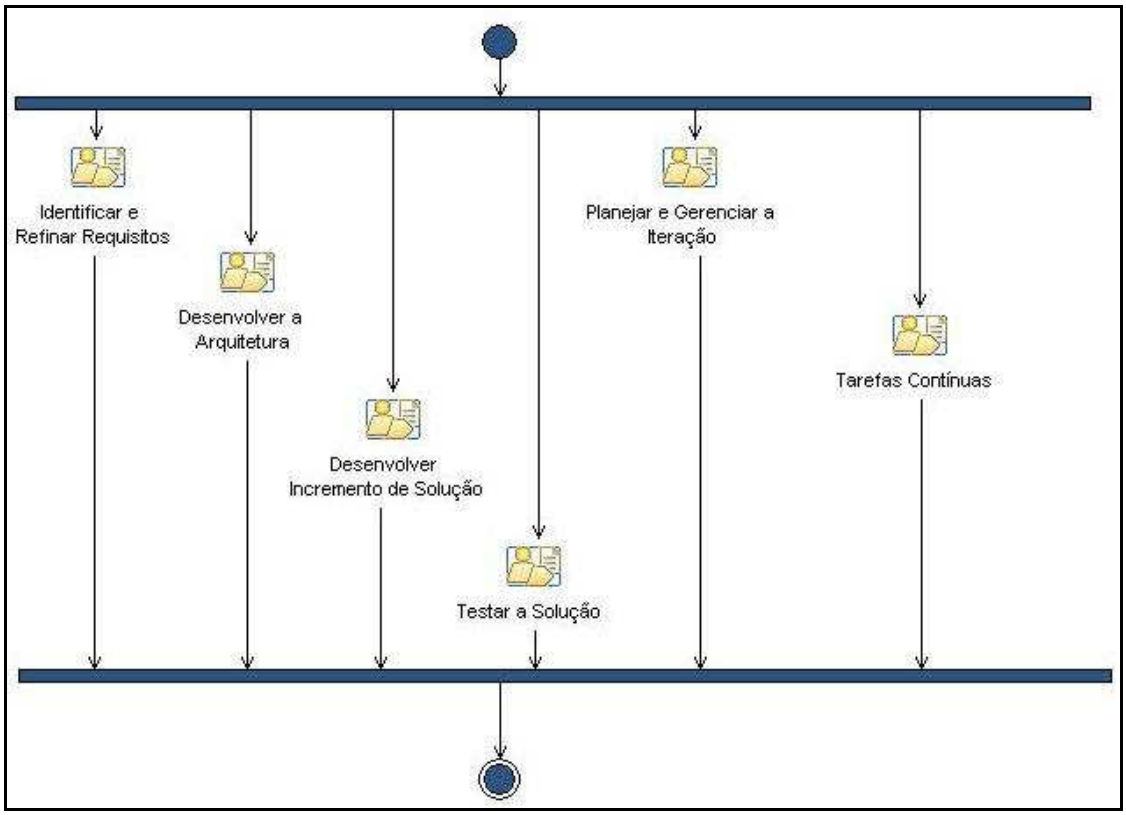
a) obter um entendimento mais detalhado dos requisitos. Um bom entendimento dos

principais requisitos permite criar um plano mais detalhado e obter comprometimento dos *stakeholders*. Ter certeza de obter um entendimento detalhado dos requisitos mais críticos que serão validados pela arquitetura;

b) projetar, implementar, validar e estabelecer uma linha de base para a arquitetura. Projetar, implementar e testar um esqueleto da estrutura do sistema. Apesar da funcionalidade não estar completa ainda, a maior parte das interfaces entre os blocos sendo construídos é implementada e testada. Isto é conhecido como uma arquitetura executável;

c) diminuir os riscos essenciais e produzir um cronograma e uma estimativa de custos precisos. Muitos riscos técnicos são resolvidos como resultado do detalhamento dos requisitos e do projeto, implementação e teste da arquitetura. Refinar e detalhar o plano de projeto de alto nível.

Para atingir os objetivos e resultados esperados durante a fase de elaboração, uma série de atividades devem ser seguidas durante as iterações, as quais são: planejar e gerenciar a iteração; identificar e refinar requisitos; desenvolver a arquitetura; desenvolver incremento de solução; testar a solução e tarefas contínuas. O fluxo de atividades da fase de elaboração pode ser observado na Figura 23.



Fonte: adaptado de Eclipse (2007b).

Figura 23 – Fluxo de atividades da fase de elaboração

No Quadro 4 pode ser observado um resumo dos objetivos da fase de elaboração e quais atividades endereçam cada objetivo.

Objetivos da fase	**Atividades que endereçam os objetivos**
- Conseguir uma compreensão mais detalhada dos requisitos. - Projetar, implementar, validar e esboçar uma Arquitetura. - Abranger os riscos essenciais e produzir cronograma e estimativa de custos precisos.	- Planejar e gerenciar a iteração. - Identificar e refinar os requisitos. - Definir a arquitetura. - Desenvolver incremento de solução. - Testar a solução. - Tarefas contínuas.

Fonte: adaptado de Eclipse (2007b).

Quadro 4 – Objetivos e atividades da fase de elaboração

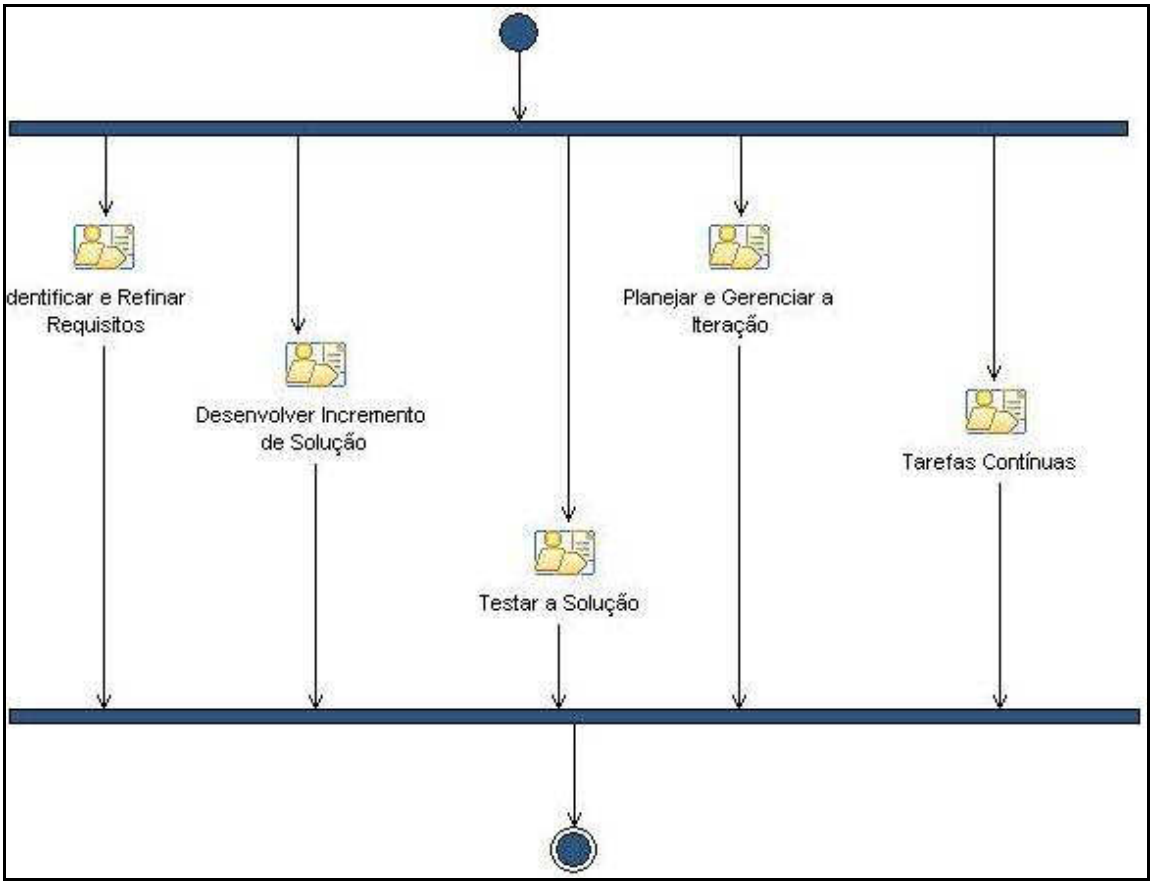
O fim da fase de elaboração é o segundo grande marco do processo, o marco da arquitetura do ciclo de vida.

A fase de construção é a terceira das quatro fases do ciclo de vida do processo. A finalidade desta fase é terminar o desenvolvimento do sistema baseado na arquitetura colocada na linha de base.

Há dois objetivos na fase de construção que ajudam a ter o desenvolvimento com custo eficiente de um produto completo, que pode ser implantado na comunidade de usuários, os quais são:

a) desenvolver de forma iterativa um produto completo que esteja pronto para ser entregue à comunidade de usuários. Descrever os requisitos restantes, preencher os detalhes do projeto, terminar a implementação e testar o software. Liberar a primeira versão operacional (beta) do sistema e determinar se os usuários já estão prontos para que a aplicação possa ser implantada;
b) minimizar os custos de desenvolvimento e conseguir algum grau de paralelismo. Otimizar os recursos e aumentar o paralelismo de desenvolvimento entre os desenvolvedores ou as equipes de desenvolvimento, atribuindo os componentes que podem ser desenvolvidos independentemente para desenvolvedores distintos.

Para atingir os objetivos e resultados esperados durante a fase de construção, uma série de atividades devem ser seguidas durante as iterações, as quais são: planejar e gerenciar a iteração; identificar e refinar requisitos; desenvolver incremento de solução; testar a solução e tarefas contínuas. O fluxo de atividades da fase de construção pode ser observado na Figura 24.



Fonte: adaptado de Eclipse (2007b).

Figura 24 – Fluxo de atividades da fase de construção

No Quadro 5 pode ser observado um resumo dos objetivos da fase de construção e quais atividades endereçam cada objetivo.

Objetivos da fase	**Atividades que endereçam os objetivos**
- Desenvolver iterativamente um produto completo que esteja pronto para a transição à sua comunidade de usuários. - Minimizar custos de desenvolvimento e alcançar algum grau de paralelismo.	- Planejar e gerenciar a iteração. - Identificar e refinar os requisitos. - Desenvolver incremento de solução. - Testar a solução. - Tarefas contínuas.

Fonte: adaptado de Eclipse (2007b).

Quadro 5 – Objetivos e atividades da fase de construção

O fim da fase de construção é o terceiro grande marco do processo, o marco da capacidade operacional inicial.

A fase de transição é a última das quatro fases do ciclo de vida do processo. A finalidade desta fase é assegurar que o software esteja pronto para ser entregue aos usuários.

Há três objetivos na fase de transição que ajudam a fazer um ajuste fino na funcionalidade, no desempenho e na qualidade total do produto beta oriundo da fase precedente, os quais são:

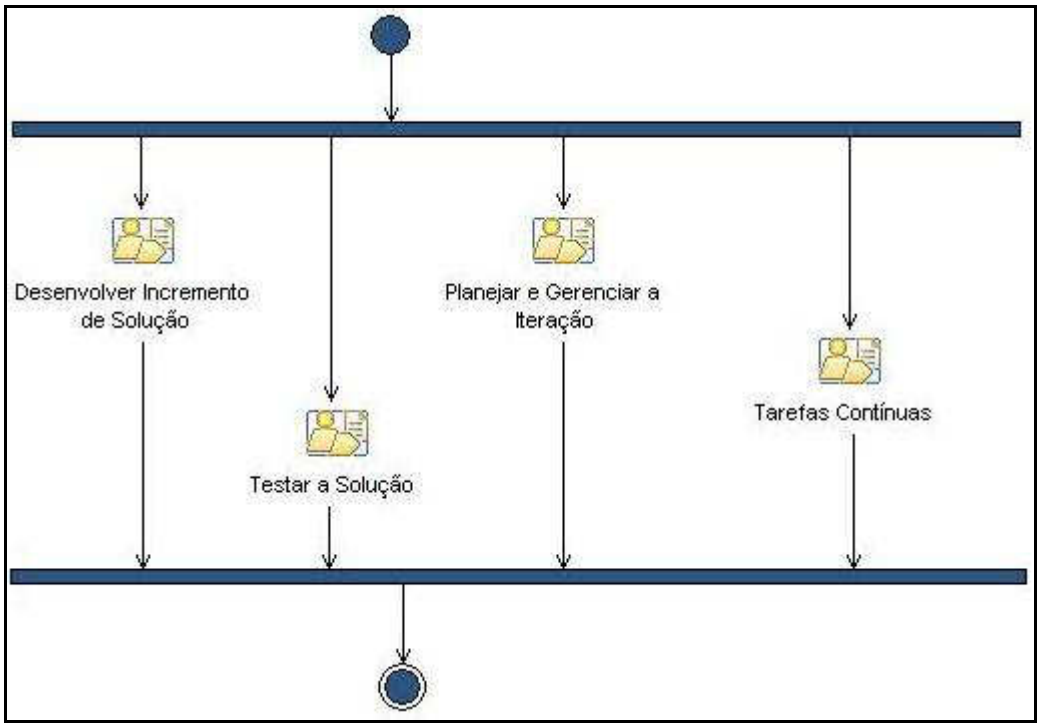
a) executar o teste beta para validar se as expectativas dos usuários foram atendidas.

Isto normalmente requer algumas atividades de ajuste fino, tais como reparação de erros e melhorias no desempenho e na usabilidade;

b) obter a concordância dos *stakeholders* de que a distribuição está completa. Isto pode envolver vários níveis de testes para a aceitação do produto, incluindo testes formais, informais e testes beta;

c) melhorar o desempenho de projetos futuros com as lições aprendidas. Documentar as lições aprendidas e melhorar o ambiente de processos e ferramentas para o projeto.

Para atingir os objetivos e resultados esperados durante a fase de transição, uma série de atividades devem ser seguidas durante as iterações, as quais são: planejar e gerenciar a iteração; desenvolver incremento de solução; testar a solução e tarefas contínuas. O fluxo de atividades da fase de transição pode ser observado na Figura 25.



Fonte: adaptado de Eclipse (2007b).

Figura 25 – Fluxo de atividades da fase de transição

No Quadro 6 pode ser observado um resumo dos objetivos da fase de transição e quais atividades endereçam cada objetivo.

Objetivos da fase	**Atividades que endereçam os objetivos**
- Realizar testes para validar que as expectativas dos usuários foram atendidas. - Obter o reconhecimento dos *stakeholders* de que o desenvolvimento esteja completo.	- Planejar e gerenciar a iteração. - Desenvolver incremento de solução. - Testar a solução. - Tarefas contínuas.

Fonte: adaptado de Eclipse (2007b).

Quadro 6 – Objetivos e atividades da fase de transição

O fim da fase de transição é o quarto grande marco do processo, o marco da liberação do produto.

3.2 ELABORAÇÃO E PUBLICAÇÃO DO PROCESSO FURBUP

Para atingir com êxito um projeto de software no ambiente acadêmico é necessário explorar um processo de desenvolvimento prático, que reúna apenas os conteúdos fundamentais para a construção de um produto de software no prazo previsto pelas disciplinas de ES da FURB. Os projetos de desenvolvimento de software propostos pelas disciplinas possuem um tempo médio de duração de quatro a oito meses com equipes variando entre três e cinco pessoas. Com isso, percebe-se que este tipo de projeto não utiliza todos os recursos do RUP e XP, dificultando o estudo de todas as atividades, conceitos e papéis que um processo de software oferece.

Com o propósito de solucionar as necessidades apontadas, decidiu-se disponibilizar um processo de desenvolvimento de software chamado FurbUP, utilizando as principais características do OpenUP que foram traduzidas para a língua portuguesa. O FurbUP segue o método iterativo e incremental de desenvolvimento, mantendo um ciclo de vida estruturado e desenvolvimento baseado em casos de uso. Assim, o FurbUP torna-se totalmente baseado no OpenUP.

Além de utilizar todos os conceitos, atividades, papéis, produtos de trabalho (Quadro 7) e ciclo de vida do OpenUP, foram realizadas algumas adaptações na passagem para o FurbUP, as quais são:

a) eliminação, adição e modificação de alguns passos para a realização de tarefas que um papel executa (anexo L);
b) inclusão do conteúdo do método de estimativa *Use Case Points*[11] (UCP);
c) inclusão de um *template* elaborado no EA, utilizando a UML nos modelos de análise mais relevantes vistos nas disciplinas de ES da FURB. Este *template* cobre boa parte dos produtos de trabalho também oferecidos pelo FurbUP.

[11] É um método de estimativa e dimensionamento de software baseado na contagem de casos de uso.

Papéis	Responsável pelos produtos de trabalho	Produtos de Trabalho de saída das tarefas que este papel desempenha
Analista	Glossário	Glossário
	Especificação de Requisitos Suplementares	Especificação de Requisitos Suplementares
	Modelo de Caso de Uso	Caso de Uso
	Visão	Lista de Itens de Trabalho
		Visão
		Modelo de Caso de Uso
Qualquer Papel		Lista de Itens de Trabalho
Arquiteto	Caderno de Arquitetura	Caderno de Arquitetura
		Design
Desenvolvedor	Construção	Construção
	Design	Design
	Implementação	Registro de Teste
	Teste de Desenvolvedor	Caso de Uso
		Lista de Itens de Trabalho
		Especificação de Requisitos Suplementares
		Testes de Desenvolvedor
		Implementação
Gerente de Projeto	Plano de Iteração	Plano de Iteração
	Lista de Itens de Trabalho	Lista de Itens de Trabalho
	Plano de Projeto	Plano de Projeto
	Lista de Riscos	Lista de Riscos
Testador	Caso de Teste	Caso de Teste
	Registro de Teste	Registro de Teste
	Script de Teste	Script de Teste
		Lista de Itens de Trabalho
Stakeholder	NENHUM PRODUTO DE TRABALHO	NENHUM PRODUTO DE TRABALHO

Quadro 7 – Relação entre os papéis e os produtos de trabalho

3.2.1 Técnicas e ferramentas utilizadas

O FurbUP foi elaborado e publicado utilizando o ambiente EPFC 1.2.0.2. Baseado nos estudos deste ambiente optou-se por construir um diagrama de classes utilizando a UML, com o objetivo de representar o relacionamento entre os seus principais componentes base. Os componentes base utilizados na elaboração do FurbUP são organizados em uma visão do tipo árvore, ou seja, uma ramificação de componentes que compõem um *plug-in*, raiz principal do processo criado pelo EPFC. Este *plug-in* FurbUP e seus respectivos componentes podem ser observados na Figura 26.

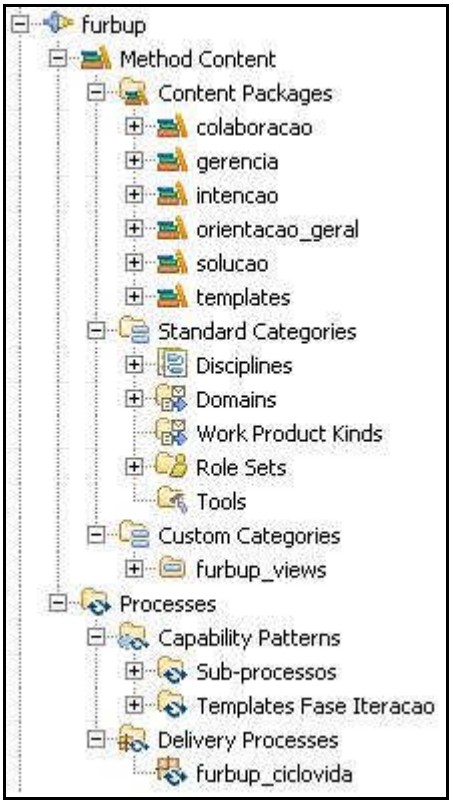


Figura 26 – *Plug-in* FurbUP e seus respectivos componentes

O diagrama de classes apresenta uma visão de como as classes estão estruturadas e relacionadas. Em função da quantidade de classes e do grande número de relacionamentos existentes, são apresentados onze diagramas, os quais são: `diagrama componentes base EPFC`; `diagrama componente categoria custom`; `diagrama componente disciplina`; `diagrama componente papel`; `diagrama componente tarefa`; `diagrama elemento conjunto papel`; `diagrama elemento domínio`; `diagrama elemento ferramenta`; `diagrama elemento orientação`; `diagrama elemento produto de trabalho` e `diagrama elemento tipo produto trabalho`.

O primeiro diagrama (Figura 27), designado de `diagrama componentes base EPFC`, contém as classes responsáveis pelos principais componentes da ferramenta EPFC que formam a base de um processo de desenvolvimento.

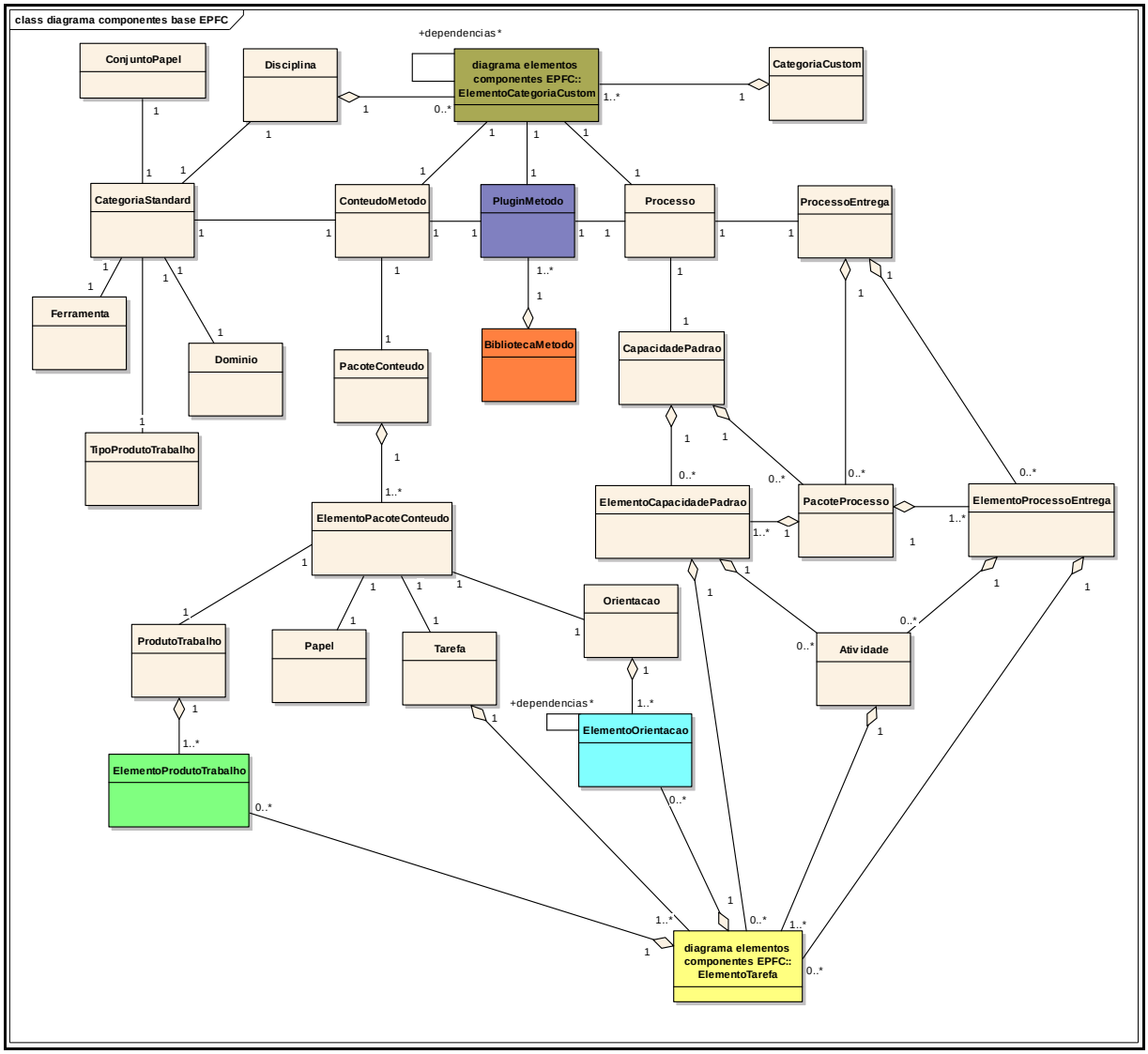


Figura 27 – Diagrama de classes `diagrama componentes base EPFC`

A classe principal deste diagrama é `BibliotecaMetodo`, que é um recipiente físico para *plug-ins* de método e definições de configurações de método. Todos os elementos de método são armazenados em uma biblioteca de métodos, o que ocorre com a classe `PluginMetodo`, que representa um recipiente físico para pacotes de método. Além disso, define o maior nível de granularidade para a modularização e organização do conteúdo de método e processos.

A classe `ConteudoMetodo` descreve conceitos metodológicos genéricos do *Unified Method Architecture*[12] (UMA) e orientações que proporcionam explicações passo a passo, descrevendo como objetivos específicos são atingidos independentemente de uma colocação destes passos dentro de um processo de ciclo de vida. O UMA separa conteúdos de método de

[12] Significa método de arquitetura unificado. UMA é uma arquitetura para a concepção, especificação, e armazenamento de metadados de métodos e processos (ECLIPSE, 2007b).

suas aplicações em processo.

A classe `Processo` é uma estrutura geral para tipos particulares de projetos de desenvolvimento. Os processos tomam elementos de conteúdo e os relacionam em seqüências semi-ordenadas que são personalizadas para tipos específicos de projetos. Assim, um processo é um conjunto de descrições de trabalho parcialmente ordenado com a intenção de alcançar uma meta de desenvolvimento mais elevada, assim como a liberação de um software específico. Estas descrições de trabalho estão organizadas em uma estrutura analítica hierárquica. Um processo foca no ciclo de vida e no seqüenciamento do trabalho em estruturas analíticas.

As classes `ProcessoEntrega` e `ElementoProcessoEntrega` são processos especiais descrevendo uma abordagem completa e integrada para executar um tipo de projeto específico. Além disso, proporcionam um modelo de ciclo de vida completo que tenha sido detalhado seqüenciando conteúdo de método em estruturas analíticas. As classes `CapacidadePadrao` e `ElementoCapacidadePadrao` também são processos especiais que descrevem uma quantidade reutilizável de uma atividade. Os padrões de capacidade expressam e comunicam conhecimento de processo para uma área de interesse chave, tal como uma disciplina e podem ser utilizados diretamente por profissionais para orientar seu trabalho. A classe `PacoteProcesso` é um pacote de método que contém somente processos como padrões de capacidade ou processos de entrega.

A classe `Atividade` é algo que um ou mais papéis fazem. No UMA, uma atividade é um elemento analítico que suporta o agrupamento lógico de elementos de processo relacionados tais como descritor e subatividades, formando assim estruturas analíticas.

A classe `CategoriaStandard` prove categorias padrões que proporcionam um meio de categorizar conteúdo de método essencial, alinhado com melhores práticas, para criar métodos estruturados. Para encorajar uma boa estrutura de método são apresentados categorias padrões para agrupar tarefas em disciplinas, produtos de trabalho em domínios, papéis em conjuntos de papéis e mentores de ferramenta em ferramentas. Diferente de categorias personalizadas, categorias padrões são por definição ligadas a tipos específicos de conteúdo de método. A classe `Disciplina` é uma coleção de tarefas relacionadas que define a principal área de interesse. Em ES, as disciplinas incluem: requisitos; arquitetura; desenvolvimento; testes e gerenciamento de projeto. A classe `ConjuntoPapel` é utilizada para agrupar papéis com certos pontos comuns. A classe `Ferramenta` é uma categoria padrão usada como recipiente para mentores de ferramentas. Também pode prover descrições gerais de ferramentas e suas

capacidades gerais. A classe `TipoProdutoTrabalho` é uma categoria padrão que representa um agrupamento de produtos de trabalho relacionados que em contraste com o domínio, é mais direcionado a apresentação, como modelos, especificações e planos. A classe `Dominio` é uma área de conhecimento ou atividade caracterizada por uma família de valores relacionados. Uma categoria de problemas específicos que é caracterizada por um corpo de conhecimentos, atividades e comportamentos, ou seja, uma hierarquia refinada que relaciona grupos de produtos de trabalho (ECLIPSE, 2007b).

A classe `PacoteConteudo` é um pacote de método especial que contém exclusivamente elementos de conteúdo. Os elementos de conteúdo são artefatos, tarefas, papéis e orientação. A classe `ElementoPacoteConteudo` é qualquer elemento modelado no UMA que seja parte do conteúdo de método. Os elementos de conteúdo proporcionam explicações passo a passo, descrevendo como metas muito específicas do desenvolvimento são alcançadas independentemente da colocação destes passos dentro do ciclo de vida do desenvolvimento. Eles são instanciados e adaptados às situações específicas dentro das estruturas de processo, o que ocorre com as classes `ProdutoTrabalho` e `ElementoProdutoTrabalho` que são elementos de conteúdo que representam qualquer coisa utilizada, produzida ou modificada por uma tarefa. A classe `Papel` é uma definição do comportamento e responsabilidades de um indivíduo ou de um conjunto de indivíduos trabalhando juntos como uma equipe. A classe `Tarefa` representa uma unidade de trabalho que um papel possa ser solicitado a executar. As classes `Orientacao` e `ElementoOrientacao` descrevem conselhos comprovados para o alcance de uma meta. No UMA, a orientação generaliza todas as formas de conteúdo cujo propósito primário seja o de proporcionar explicações sobre outros elementos do UMA. A orientação sendo ela mesma um elemento de conteúdo, torna possível associar orientação à outra orientação.

As classes `CategoriaCustom`, `ElementoCategoriaCustom` e `ElementoTarefa` serão descritas nos diagramas `diagrama componente categoria custom` e `diagrama componente tarefa`.

O segundo diagrama (Figura 28), designado de `diagrama componente categoria custom`, contém as classes responsáveis pela estruturação do processo na visão do usuário.

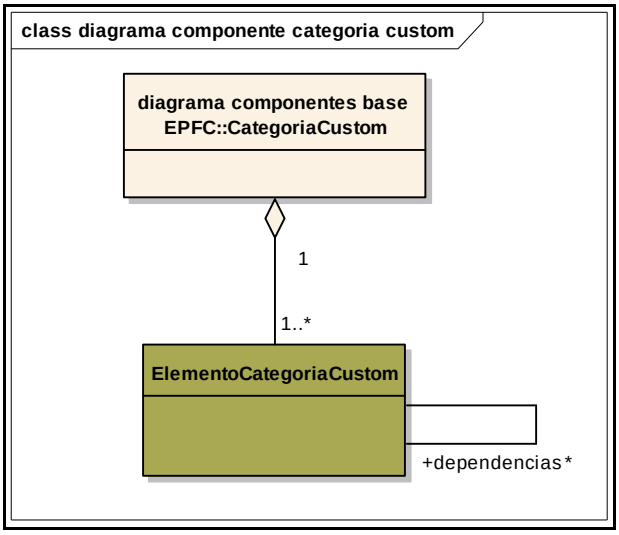


Figura 28 – Diagrama de classes `diagrama componente categoria custom`

As classes `CategoriaCustom` e `ElementoCategoriaCustom` são responsáveis por classificar o conteúdo do processo baseado nos critérios do usuário. Um uso importante é na construção de visões.

O terceiro diagrama (Figura 29), designado de `diagrama componente disciplina`, contém as classes responsáveis pelo relacionamento entre uma coleção de tarefas que definem uma área de interesse.

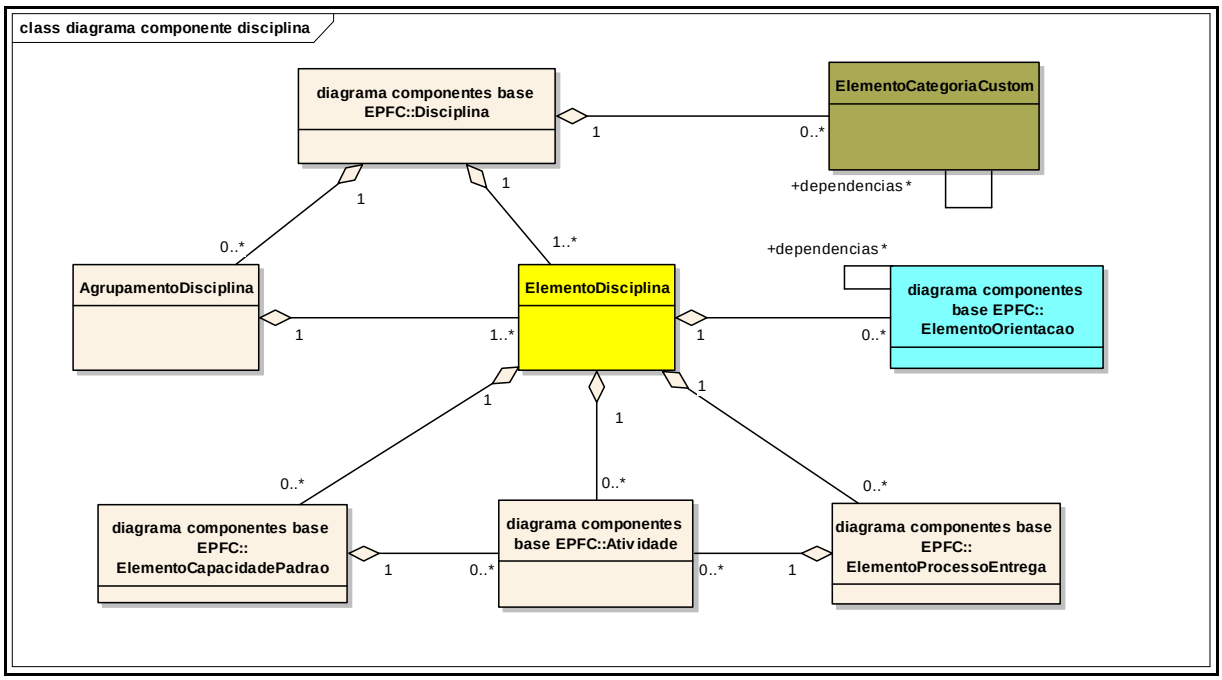


Figura 29 – Diagrama de classes `diagrama componente disciplina`

As classes `Disciplina`, `ElementoCategoriaCustom`, `ElementoOrientacao`, `ElementoProcessoEntrega`, `Atividade` e `ElementoCapacidadePadrao` são idênticas às classes apresentadas nos diagramas `diagrama componentes base EPFC` e `diagrama componente categoria custom`. A classe `ElementoDisciplina`, assim como a classe

`Disciplina`, é uma coleção de tarefas relacionadas que define a principal área de interesse em um processo de desenvolvimento. A classe `AgrupamentoDisciplina` é utilizada para agrupar disciplinas com certos pontos comuns no processo (ECLIPSE, 2007b).

O quarto diagrama (Figura 30), designado de `diagrama componente papel`, contém as classes responsáveis pelo relacionamento de componentes base a um comportamento definido e direcionado a um ou mais indivíduos que podem formar uma equipe de trabalho.

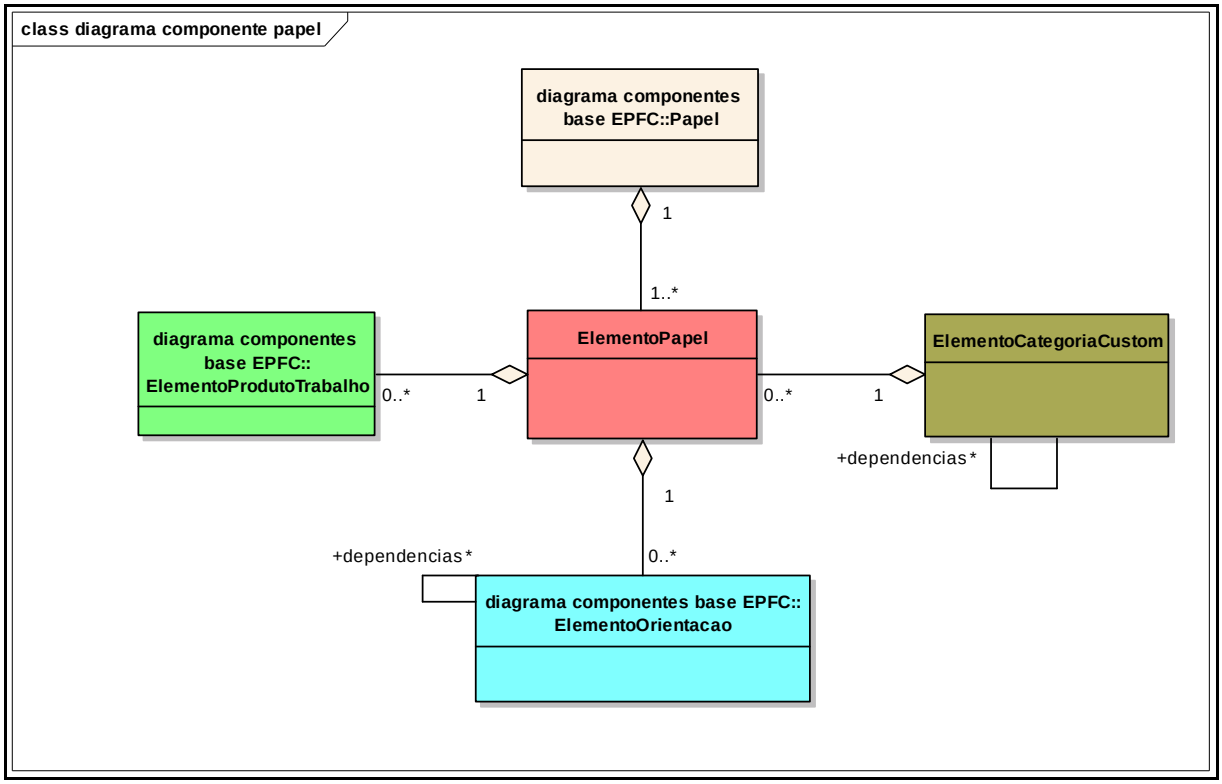


Figura 30 – Diagrama de classes `diagrama componente papel`

As classes `Papel`, `ElementoProdutoTrabalho`, `ElementoOrientacao` e `ElementoCategoriaCustom` são idênticas às classes apresentadas nos diagramas `diagrama componentes base EPFC` e `diagrama componente categoria custom`. A classe `ElementoPapel`, assim como a classe `Papel`, é uma definição do comportamento e responsabilidades que um ou mais indivíduos devem realizar em um processo de desenvolvimento.

O quinto diagrama (Figura 31), designado de `diagrama componente tarefa`, contém as classes responsáveis pelo relacionamento de componentes base a uma tarefa que um papel é solicitado a executar.

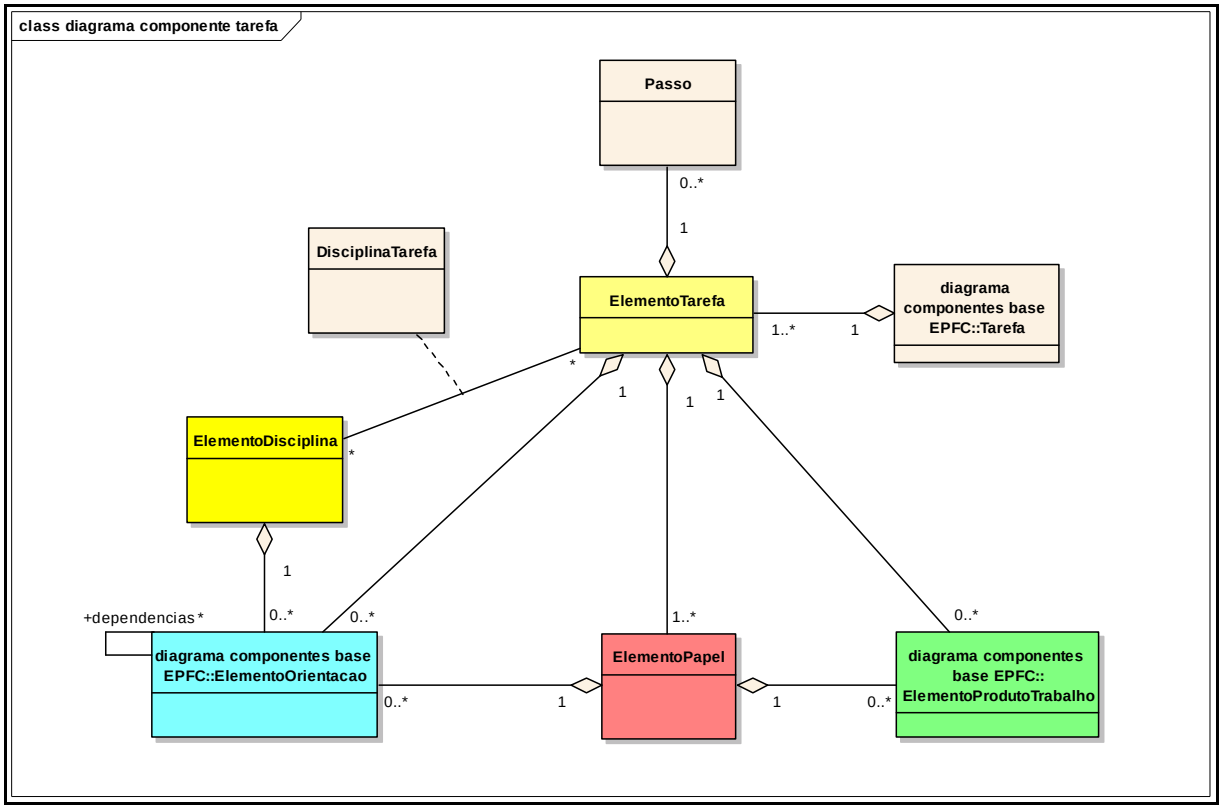


Figura 31 – Diagrama de classes `diagrama componente tarefa`

As classes `Tarefa`, `ElementoProdutoTrabalho`, `ElementoPapel`, `ElementoOrientacao` e `ElementoDisciplina` são idênticas às classes apresentadas nos diagramas `diagrama componentes base EPFC`, `diagrama componente disciplina` e `diagrama componente papel`. A classe `ElementoTarefa`, assim como a classe `Tarefa`, é uma unidade de trabalho que um papel possa ser designado a executar. A classe `Passo`, no UMA, é um elemento de conteúdo utilizado para organizar tarefas em partes ou subunidades de trabalho. A classe `DisciplinaTarefa`, representa a associação de elementos de disciplinas com elementos de tarefa.

O sexto diagrama (Figura 32), designado de `diagrama elemento conjunto papel`, contém as classes responsáveis pelo relacionamento de componentes base a um conjunto de papéis utilizados para agrupar papéis que possuem pontos em comum.

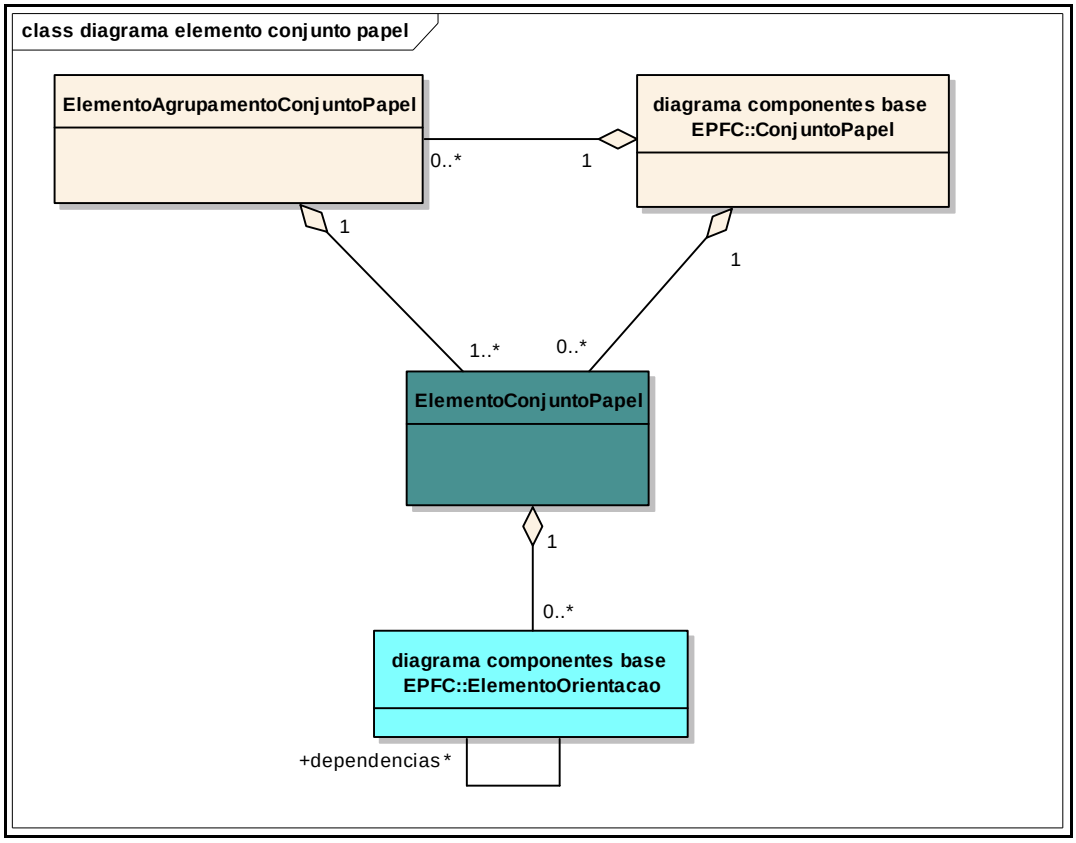


Figura 32 – Diagrama de classes `diagrama elemento conjunto papel`

As classes `ConjuntoPapel` e `ElementoOrientacao` são idênticas às apresentadas no diagrama `diagrama componentes base EPFC`. A classe `ElementoConjuntoPapel`, assim como a classe `ConjuntoPapel`, é utilizada para agrupar papéis que possuam similaridades. A classe `ElementoAgrupamentoConjuntoPapel` é utilizada para agrupar grupos de papéis com certos pontos comuns.

O sétimo diagrama (Figura 33), designado de `diagrama elemento domínio`, contém as classes responsáveis pelo relacionamento de componentes base a uma classe de problemas específicos que é caracterizada por um conjunto de conhecimentos, atividades e comportamentos.

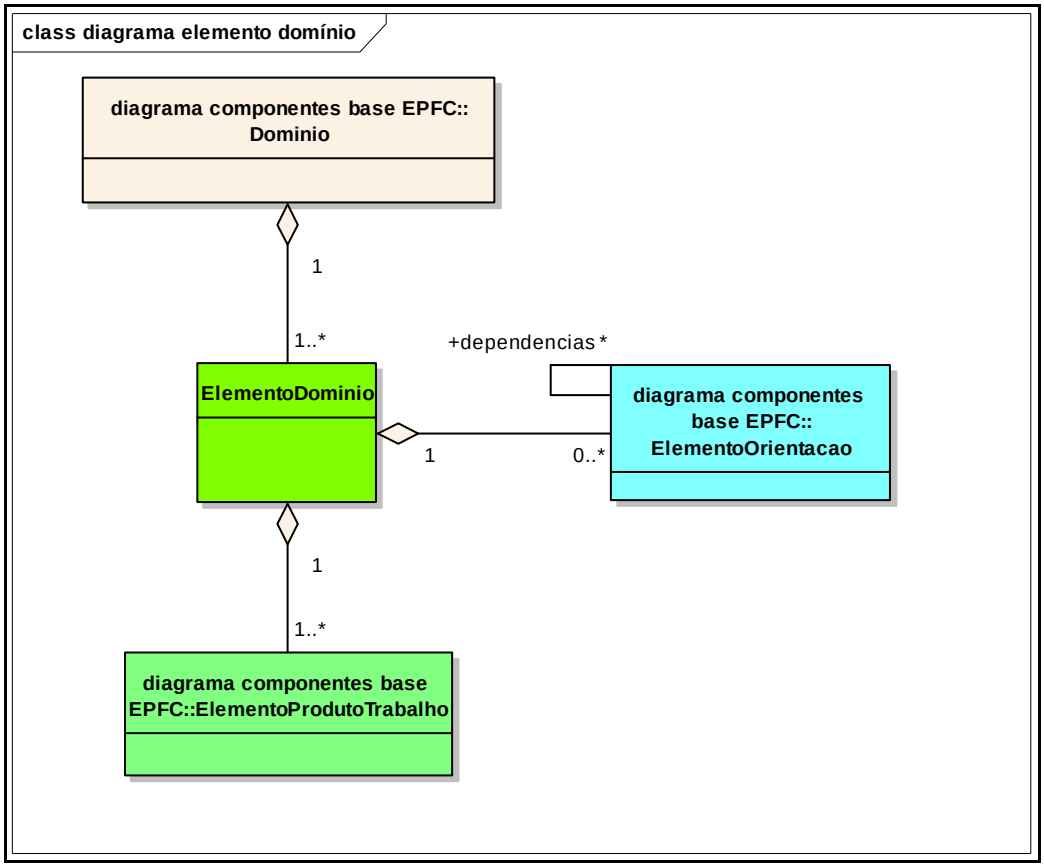


Figura 33 – Diagrama de classes `diagrama elemento domínio`

As classes `Dominio`, `ElementoOrientacao` e `ElementoProdutoTrabalho` são idênticas às apresentadas no diagrama `diagrama componentes base EPFC`. A classe `ElementoDominio`, assim como a classe `Dominio`, é uma ordem de subordinação refinada que relaciona grupos de produtos de trabalho.

O oitavo diagrama (Figura 34), designado de `diagrama elemento ferramenta`, contém as classes responsáveis pelo relacionamento de componentes base a descrições gerais das principais características de ferramentas e suas habilidades.

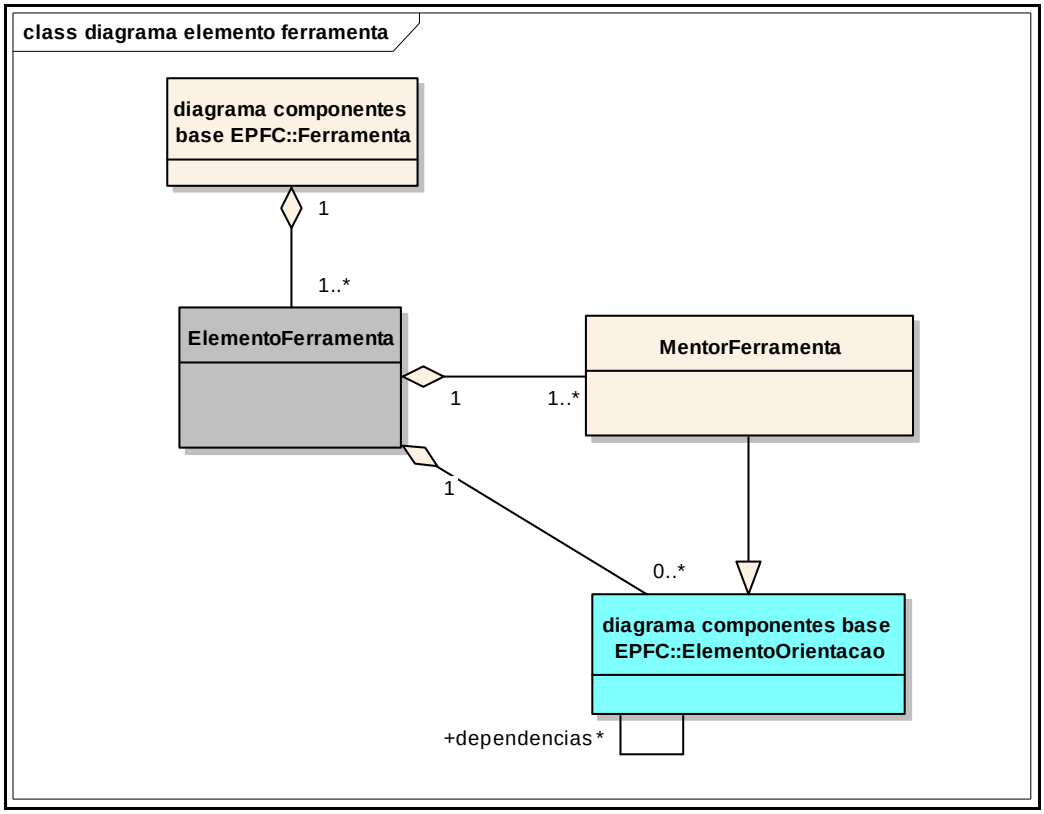


Figura 34 – Diagrama de classes `diagrama elemento ferramenta`

As classes `Ferramenta` e `ElementoOrientacao` são idênticas às apresentadas no diagrama `diagrama componentes base EPFC`. A classe `ElementoFerramenta`, assim como a classe `Ferramenta`, é um objeto capaz de conter mentores de ferramentas. A classe `MentorFerramenta` é um tipo de orientação que descreve como realizar tarefas utilizando um ambiente de software específico.

O nono diagrama (Figura 35), designado de `diagrama elemento orientação`, contém as classes responsáveis pelos componentes de orientação do EPFC. Esses componentes descrevem pareceres comprovados para que certos objetivos sejam alcançados em um processo de desenvolvimento.

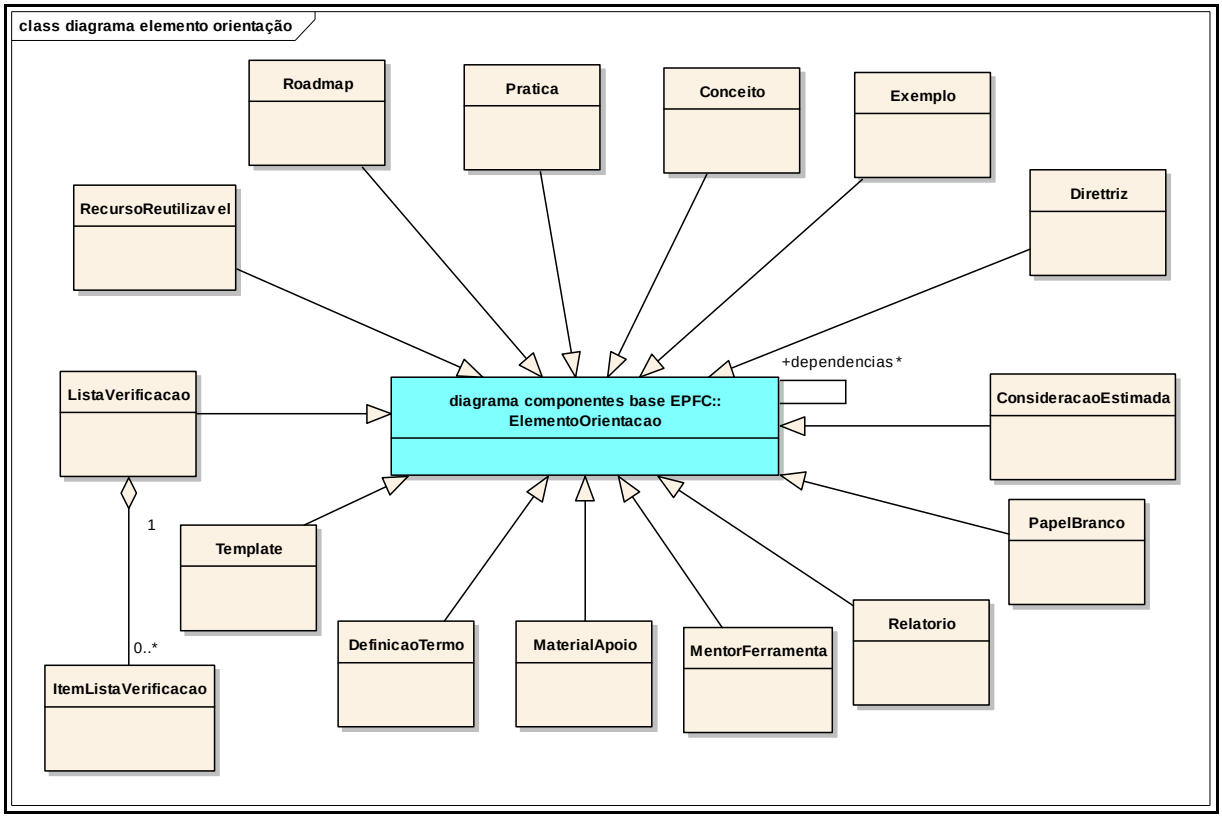


Figura 35 – Diagrama de classes `diagrama elemento orientação`

As classes `ElementoOrientacao` e `MentorFerramenta` são idênticas às apresentadas nos diagramas `diagrama componentes base EPFC` e `diagrama elemento ferramenta`. A classe `RecursoReutilizavel` é uma orientação que descreve um determinado recurso como código fonte, *templates*, padrões, estruturas arquiteturais e modelos de domínio que pode ser reutilizado em um diferente contexto. A classe `Roadmap` é uma orientação que resume um processo, com uma perspectiva particular, como ao promover uma passagem passo a passo ou uma típica instanciação de atividades. A classe `Pratica` é uma orientação que apresenta uma maneira ou estratégia comprovada de realizar trabalho para alcançar uma meta que tenha um impacto positivo na qualidade do produto de trabalho ou processo. A classe `Conceito` esboça idéias chave ou princípios básicos subjacentes a um tópico central ao método. Os conceitos normalmente se referem a tópicos mais genéricos do que as diretrizes e se estendem por diversos produtos de trabalho, tarefas ou atividades. A classe `Diretriz` é uma orientação que proporciona detalhes adicionais de como lidar com um elemento de conteúdo em particular. As diretrizes se aplicam mais freqüentemente a tarefas e a produtos de trabalho. A classe `Exemplo` é uma orientação que representa uma instância de amostra típica e parcialmente completa de um ou mais elementos de conteúdo. Os exemplos são fornecidos mais freqüentemente para produtos de trabalho. A classe `ConsideracaoEstimada` é um tipo específico de orientação que proporciona medidas de dimensionamento, ou padrões para

dimensionar o esforço de trabalho associado com a execução de uma parcela particular de trabalho, bem como instruções para o seu uso com sucesso. Pode conter considerações de estimação e métricas de estimação. A classe `PapelBranco` é um tipo de orientação para papéis publicados externamente que podem ser lidos e compreendidos isolados de outros elementos de conteúdo e orientações. A classe `Relatorio` é uma orientação com um *template* predefinido de um resultado baseado em outro produto de trabalho. Uma saída de algum formulário ou ferramenta de automação. A classe `MaterialApoio` é uma orientação que possui uma gama de possibilidades para outros tipos de orientações não definidas especificamente em outro lugar no processo. A classe `DefinicaoTermo` é uma orientação que define conceitos que são utilizados para construir o glossário do processo. A classe `Template` é uma orientação que especifica a estrutura de um produto de trabalho ao proporcionar uma tabela de conteúdos predefinida, seções, pacotes, e/ou títulos, um formato padronizado, bem como descrições de como as seções e pacotes deveriam ser utilizados e completados. As classes `ListaVerificacao` e `ItemListaVerificacao` identificam uma série de itens que precisam ser completados ou verificados. Listas de verificação são utilizadas freqüentemente em revisões, passagens passo a passo ou inspeções (anexo M) (ECLIPSE, 2007b).

O décimo diagrama (Figura 36), designado de `diagrama elemento produto de trabalho`, contém as classes responsáveis pelo relacionamento de componentes base a elementos de conteúdo que representam qualquer artefato utilizado, produzido ou modificado pela execução de uma tarefa.

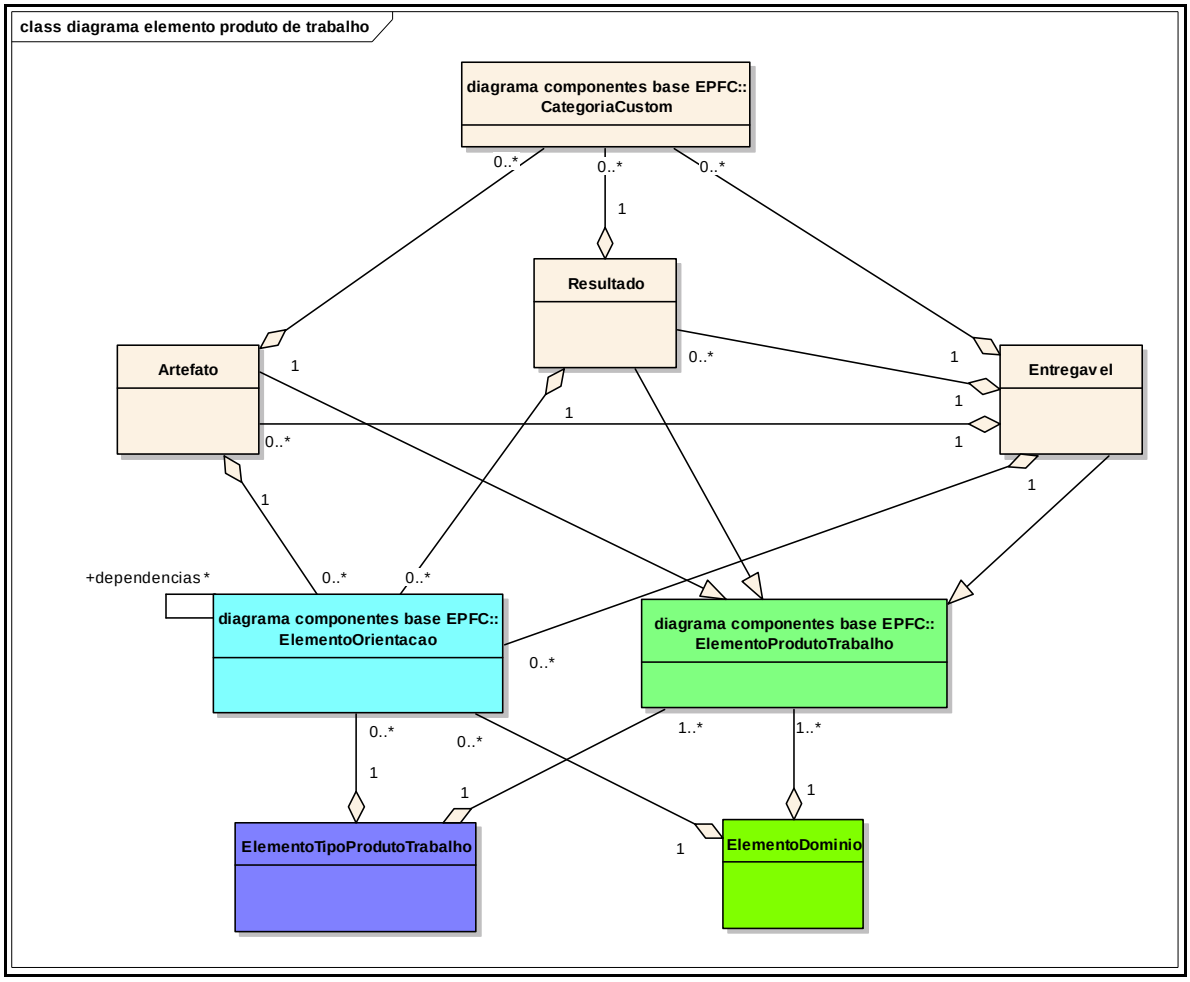


Figura 36 – Diagrama de classes `diagrama elemento produto de trabalho`

As classes `ElementoProdutoTrabalho`, `ElementoOrientacao`, `CategoriaCustom` e `ElementoDominio` são idênticas às apresentadas nos diagramas `diagrama componentes base EPFC` e `diagrama elemento domínio`. A classe `Artefato` é um produto de trabalho formal que é produzido, modificado ou utilizado por uma tarefa. Define uma área de responsabilidade e está sujeito a controle de versão. Um artefato pode ter múltiplas formas incluindo um modelo, um elemento modelo ou um documento. A classe `Resultado` descreve produtos de trabalho intangíveis que são um resultado ou um estado. Um resultado pode também ser usado para representar um produto de trabalho informal. A classe `Entregavel` é uma saída de um processo que tenha um valor material ou outro, para um cliente ou outro *stakeholder* (ECLIPSE, 2007b).

A classe `ElementoTipoProdutoTrabalho` será descrita no diagrama `diagrama elemento tipo produto trabalho`.

O décimo primeiro diagrama (Figura 37), designado de `diagrama elemento tipo produto trabalho`, contém as classes responsáveis pelo relacionamento de componentes base ao agrupamento de produtos de trabalhos que se relacionam entre si.

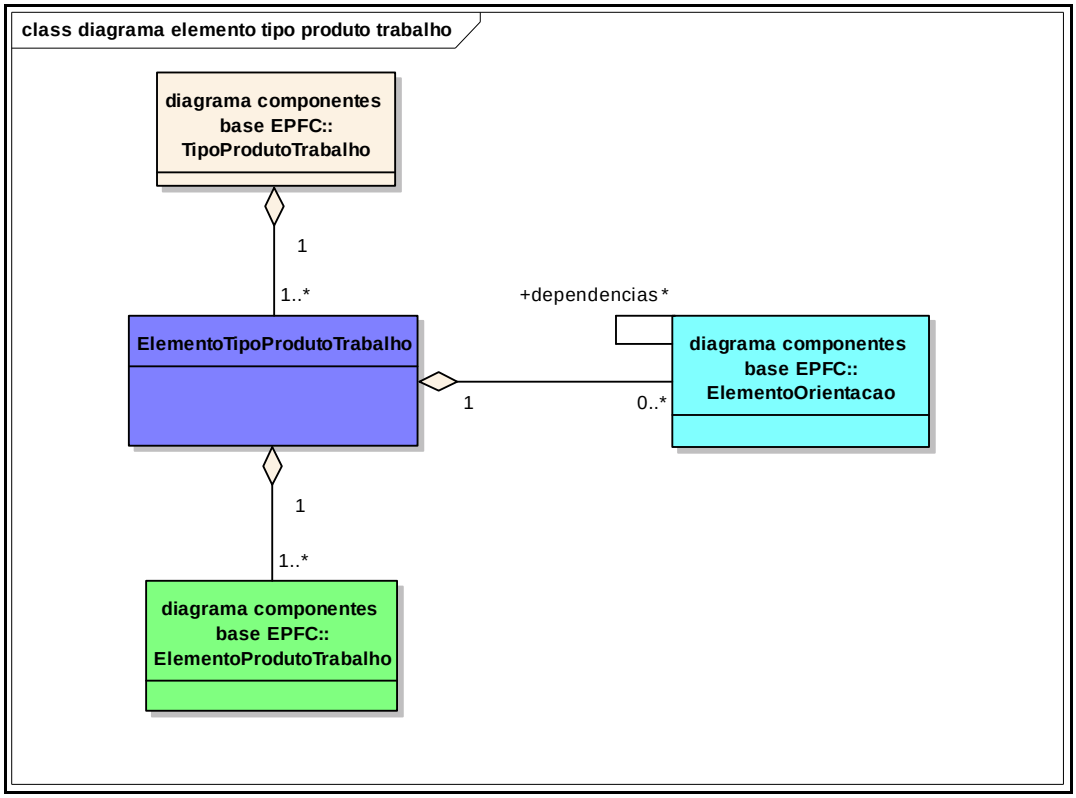


Figura 37 – Diagrama de classes `diagrama elemento tipo produto trabalho`

As classes `TipoProdutoTrabalho`, `ElementoOrientacao` e `ElementoProdutoTrabalho` são idênticas às apresentadas no diagrama `diagrama componentes base EPFC`. A classe `ElementoTipoProdutoTrabalho`, assim como a classe `TipoProdutoTrabalho`, é uma categoria padrão que representa o ajuntamento de produtos de trabalho relacionados com relação ao determinado domínio.

3.2.2 Operacionalidade do processo

Nesta seção é apresentada a operacionalidade do processo FurbUP. O processo é apresentado na Figura 38, possuindo algumas informações básicas e *links* para acesso rápido ao conteúdo relacionado à introdução, disciplinas, produtos de trabalho, papéis, ciclo de vida, entre outros. Os elementos do conteúdo do método estão organizados em navegadores à esquerda, no formato de uma árvore, disponibilizando ao usuário um acesso estruturado às várias visões do processo.

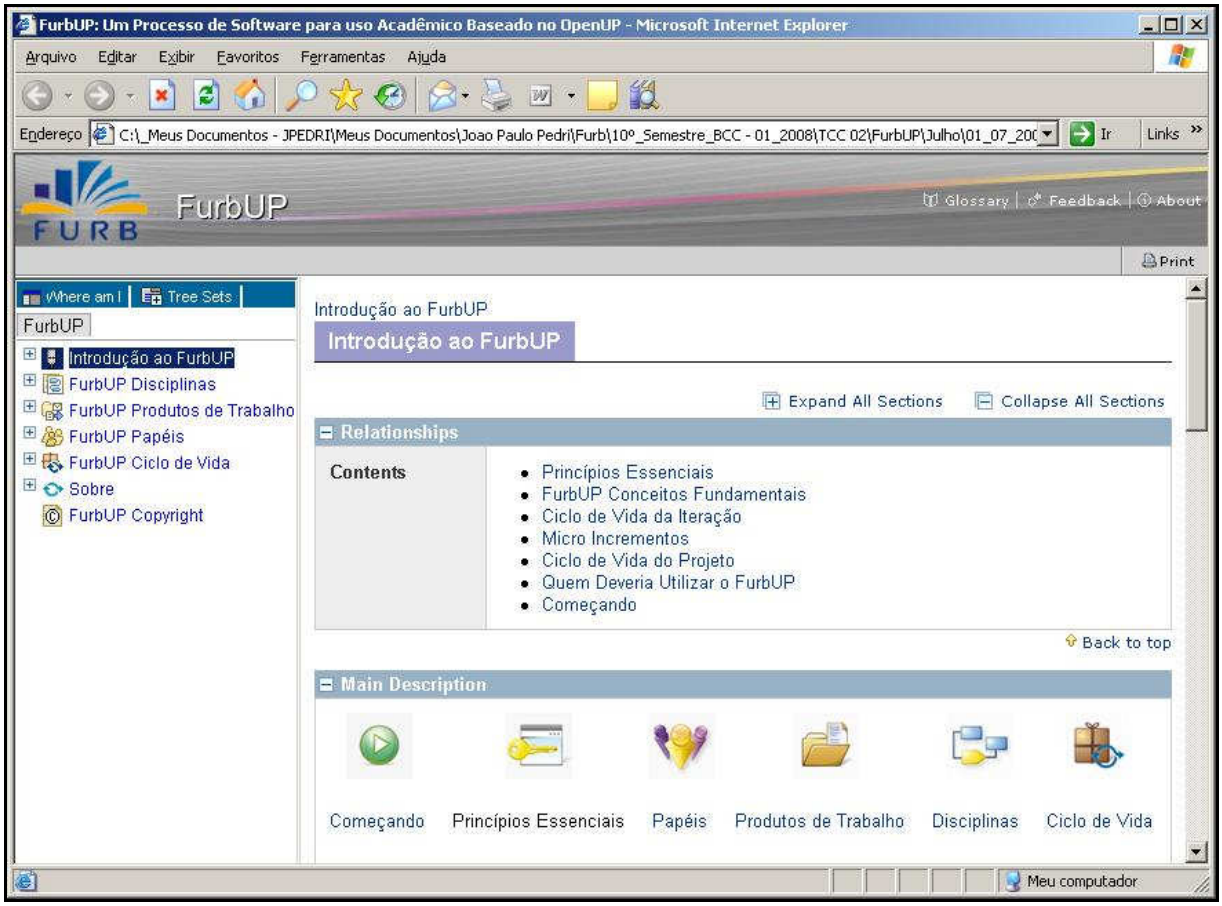


Figura 38 – Interface principal do FurbUP

Através do *link* `Glossary`, é possível acessar a interface que permite a navegação entre os itens do glossário do processo (Figura 39).

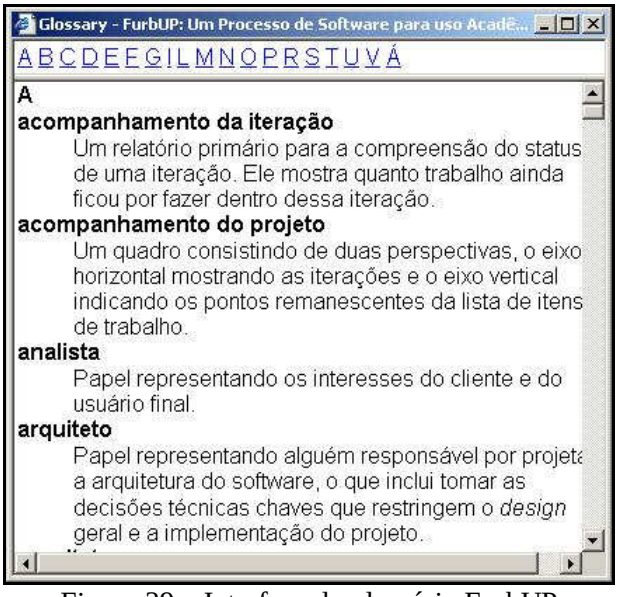


Figura 39 – Interface do glossário FurbUP

Ainda na Figura 38, através do *link* `Feedback`, é possível enviar uma mensagem pelo correio eletrônico à equipe de desenvolvimento do processo, caso o usuário encontre um erro ou até mesmo dicas e sugestões para melhorar o processo. No *link* `Introdução ao FurbUP`, o

usuário pode acessar os princípios essenciais, conceitos fundamentais, entre outros conteúdos do processo (Figura 40).

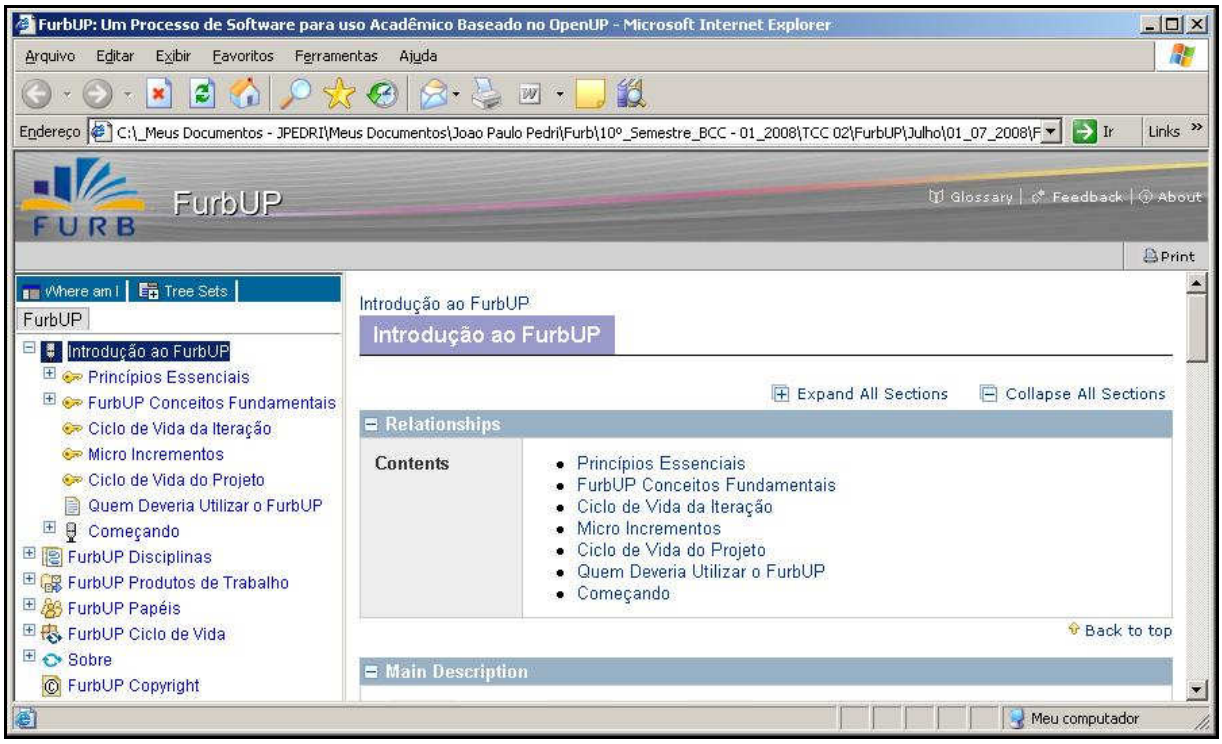


Figura 40 – Interface da introdução ao FurbUP

Através do *link* `FurbUP Disciplinas`, é possível acessar os conteúdos relacionados à cada disciplina do processo, como suas tarefas e conceitos (Figura 41).

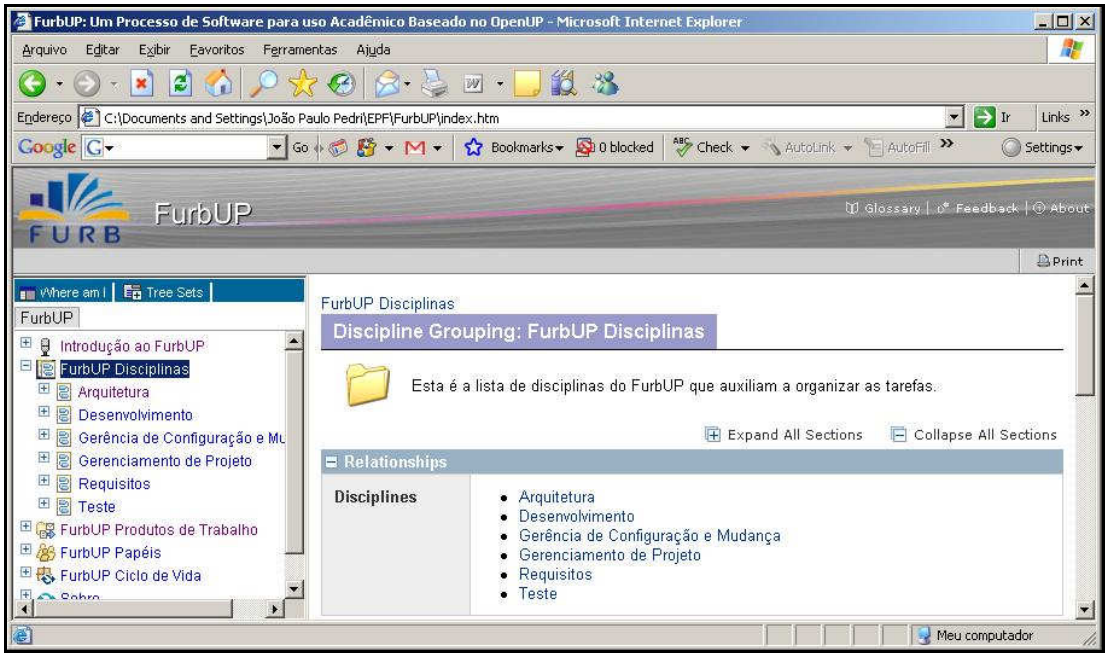


Figura 41 – Interface das disciplinas FurbUP

No *link* `FurbUP Produtos de Trabalho`, o usuário pode acessar, através de domínios, os produtos de trabalho alocados ao processo e seus eventuais *templates* (Figura 42).

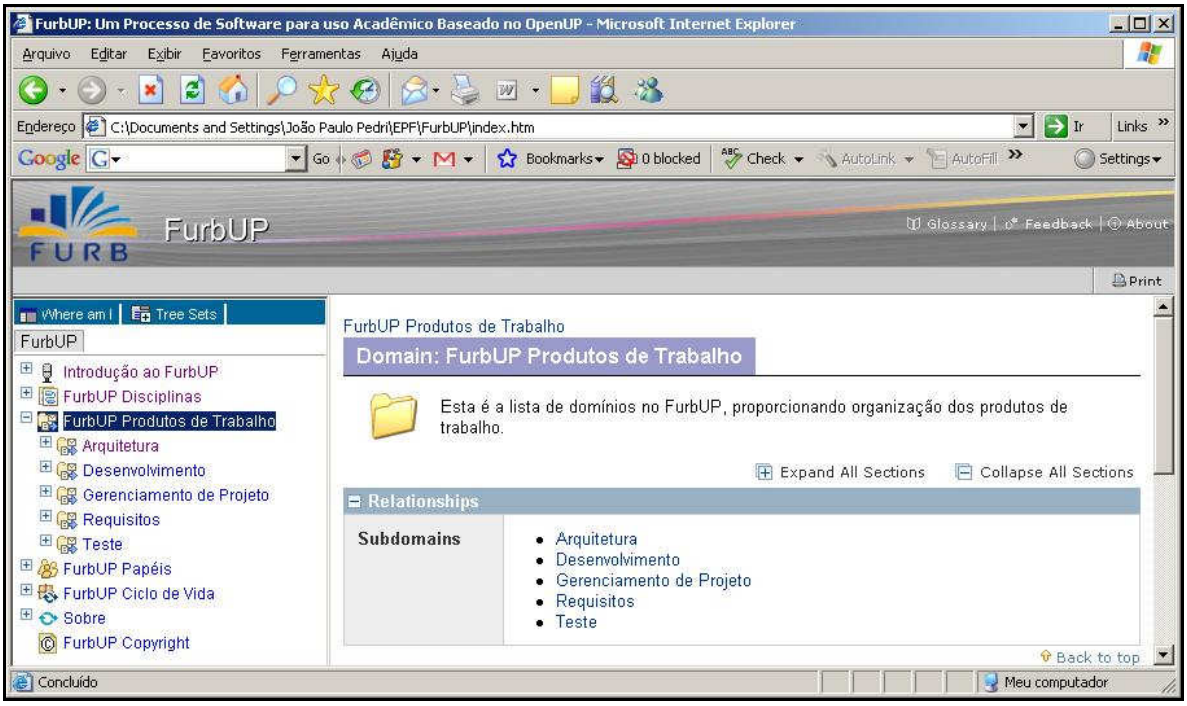


Figura 42 – Interface dos domínios FurbUP

Através do *link* `FurbUP Papéis`, é possível acessar os conteúdos dos principais papéis do processo, como as responsabilidades em executar tarefas e produzir ou modificar artefatos (Figura 43).

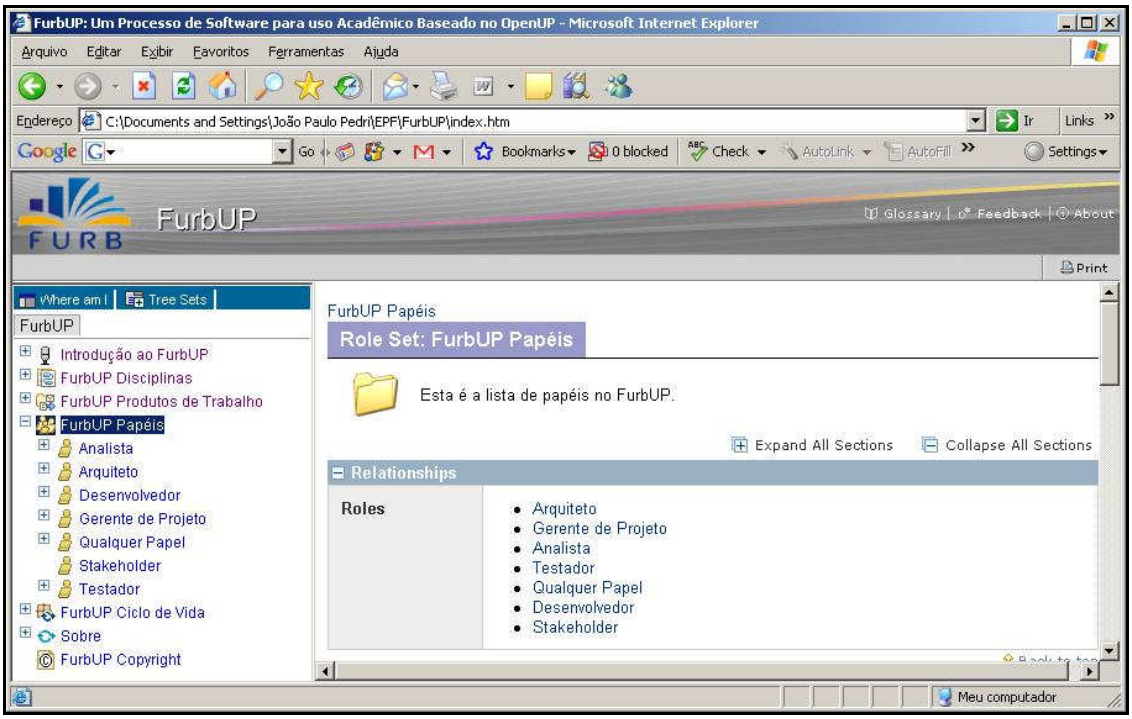


Figura 43 – Interface dos papéis FurbUP

No *link* `FurbUP Ciclo de Vida`, o usuário pode acessar, através de quatro fases de iteração, as atividades que os papéis devem executar para um certo número de iterações estabelecidas de acordo com o projeto (Figura 44). Além disso, é possível acessar os marcos e

conceitos de cada uma das quatro fases do processo.

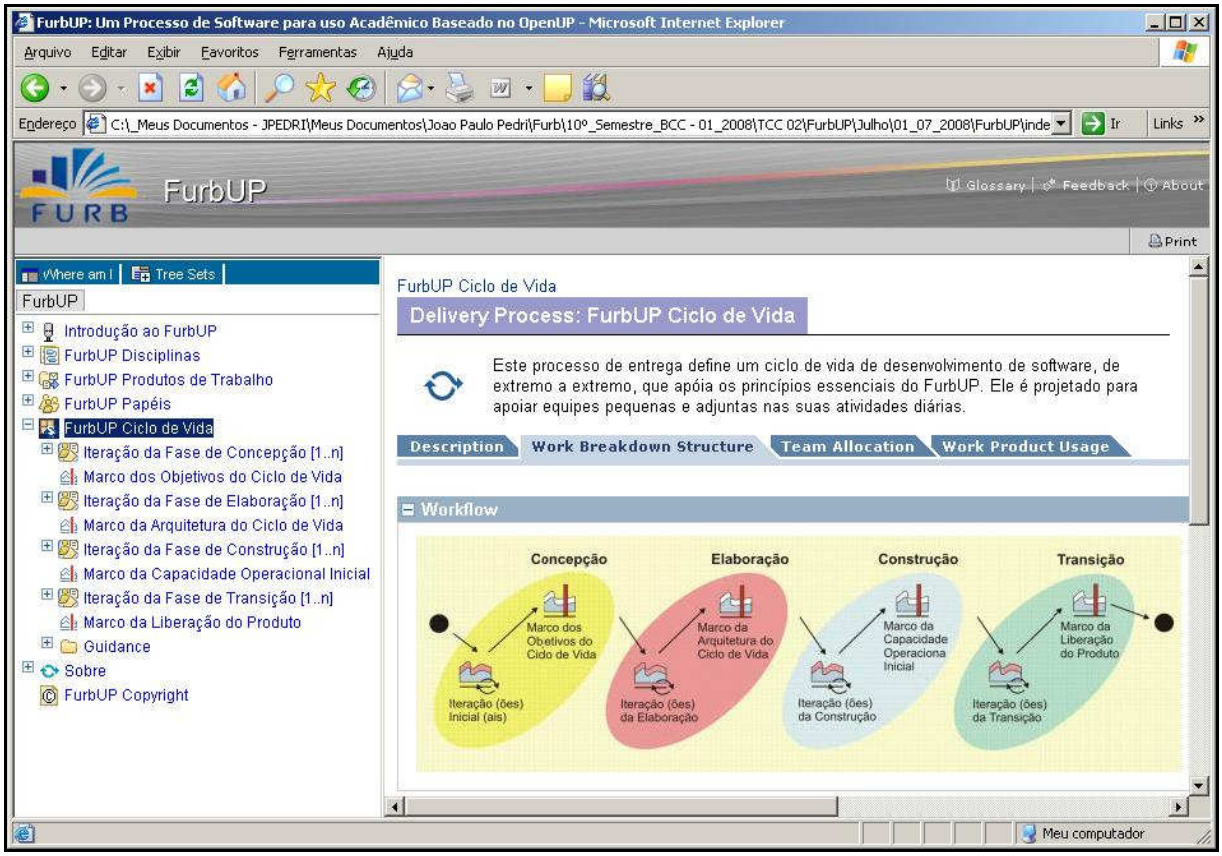


Figura 44 – Interface ciclo de vida FurbUP

3.3 DESENVOLVIMENTO DO ESTUDO DE CASO

Com o objetivo de explorar as principais características do FurbUP ao longo das disciplinas da área de ES da FURB, optou-se por disponibilizar um estudo de caso, representado através de um sistema de reserva de laboratórios. Este estudo de caso visa apresentar ao acadêmico quais são as etapas necessárias para a elaboração de um produto de software utilizando conceitos e práticas do processo FurbUP.

3.3.1 Requisitos do sistema

Para validar o processo de desenvolvimento publicado, optou-se por implementar um

sistema de reserva de laboratórios que foi utilizado como estudo de caso. Este estudo de caso procurou atender as principais tarefas, disciplinas e artefatos do processo de software FurbUP, tendo como resultado final um produto de software e sua documentação associada.

O estudo de caso possui o seguinte relato (Quadro 8).

Baseado na experiência em diversas universidades verificou-se que professores usam os laboratórios de informática para ministrar algumas de suas aulas. Mas, para utilizar um laboratório, eles devem solicitar ao coordenador do curso que reserve um laboratório para seu horário. Ao reservar um laboratório, o coordenador leva em consideração outros fatores além da disponibilidade de um laboratório em determinado horário. É verificado se o laboratório já possui instalados os softwares que serão utilizados pela disciplina. Além disso, os laboratórios com maior número de computadores são destinados preferencialmente às turmas com maior número de alunos. Há também disciplinas que possuem horários fixos em laboratórios, ou seja, as aulas são totalmente feitas em laboratório durante todo o semestre. Isto significa que determinados laboratórios em determinados horários já estão reservados para o professor. O coordenador também mantém registros destas disciplinas fixas em laboratórios para não reservar estes laboratórios para outros professores. Quando ocorre conflito, ou seja, um professor deseja reservar um laboratório já reservado por outro professor, este deve negociar com o professor para disponibilizar o laboratório. Se estes entraram em acordo e o professor ceder à reserva, o coordenador deve ser comunicado para atualizar seus registros. Muitas universidades utilizam vários meios de controlar as reservas de laboratórios. Uma delas é utilizando uma planilha eletrônica, a qual o coordenador mantém todos os registros de reservas efetuadas. A planilha contempla várias informações como o dia da reserva, o horário, nome do professor, disciplina e laboratório reservado. Através desta planilha é realizado todo o controle de reservas de laboratório, o que torna o processo lento, pois os professores precisam contatar o coordenador do curso para reservar um laboratório. Esta planilha de reservas fica disponível ao funcionário que controla as chaves e acessos dos laboratórios. O funcionário, antes de entregar a chave ao professor, verifica na planilha se este reservou o laboratório, e então libera o laboratório para uso do professor. O sistema automatizará todo este processo de reserva de laboratório. Fornecerá ao professor o controle das suas reservas e eliminará a presença do coordenador para efetuar a reserva. Agilizará o processo e reduzirá o esforço do coordenador. O professor poderá reservar um laboratório a qualquer hora e o funcionário receberá o registro atualizado das reservas instantaneamente, através do relatório de reservas, disponível ao funcionário e aos professores.

Quadro 8 – Descrição do estudo de caso

Com base nas informações descritas, são apresentados os requisitos funcionais e não funcionais atendidos pelo sistema de reserva de laboratórios. No Quadro 9 podem ser observados os requisitos funcionais. Já no Quadro 10, podem ser observados os requisitos não funcionais.

REQUISITOS FUNCIONAIS	CASO DE USO
RF01 - O sistema deve efetuar as reservas de laboratórios para as disciplinas.	UC01
RF02 - O sistema deve permitir que o professor verifique quais os conflitos existentes, ou seja, quais professores já reservaram um laboratório em um determinado horário.	UC01
RF03 - O sistema deve cadastrar os softwares utilizados em cada disciplina de cada professor.	UC02
RF04 - O sistema deve apresentar a lista das reservas efetuadas, apresentando a data, horário e para quem está reservado o laboratório.	UC03
RF05 - O sistema deve cadastrar os professores.	UC05
RF06 - O sistema deve cadastrar as disciplinas dos professores, assim como os respectivos horários das aulas.	UC06 UC12
RF07 - O sistema deve efetuar o registro de reservas fixas (reservas para disciplinas 100% práticas - em laboratório).	UC07
RF08 - O sistema deve cadastrar os laboratórios.	UC09
RF09 - O sistema deve cadastrar os softwares instalados em cada laboratório.	UC10
RF10 - O sistema deve cadastrar os softwares que são utilizados na universidade.	UC08
RF11 - O sistema deve cadastrar os usuários administrativos.	UC11

Quadro 9 – Requisitos funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01 - O sistema deve utilizar senhas de acesso para o controle seguro da aplicação.
RNF02 - O sistema deve ser desenvolvido na linguagem Java.
RNF03 - O sistema deve ser desenvolvido utilizando o ambiente NetBeans 5.5 IDE, seguindo o modelo de arquitetura em camadas conhecida como *Model, View* e *Controller*[13] (MVC).
RNF04 - O sistema deve ser desenvolvido na plataforma Windows XP.
RNF05 - O sistema deve utilizar o banco de dados MySQL na versão 4.1 em diante.

Quadro 10 – Requisitos não funcionais

3.3.2 Especificação do sistema

O sistema de reserva de laboratórios foi especificado através da ferramenta Enterprise Architect (EA), utilizando os conceitos de orientação a objetos e baseando-se nos diagramas da UML, gerando como artefatos neste texto o diagrama de casos de uso, suas descrições e o diagrama de classes. Além disso, é apresentado o diagrama de entidade relacionamento. Outros diagramas foram construídos para este sistema, porém estão disponíveis no *template* e anexo ao CD deste trabalho.

Nas subseções a seguir são apresentados os diagramas de casos de uso, classes (visão lógica) e entidade relacionamento que contemplam a especificação do sistema.

[13] MVC é um modelo de desenvolvimento em camadas que tem o objetivo de dividir as funcionalidades de um sistema.

3.3.2.1 Casos de uso

O sistema de reserva de laboratórios possui doze casos de uso (Figura 45). Estes casos de uso estão distribuídos em três áreas de acesso, os quais são:

a) área restrita ao professor;
b) área pública;
c) área restrita ao administrador.

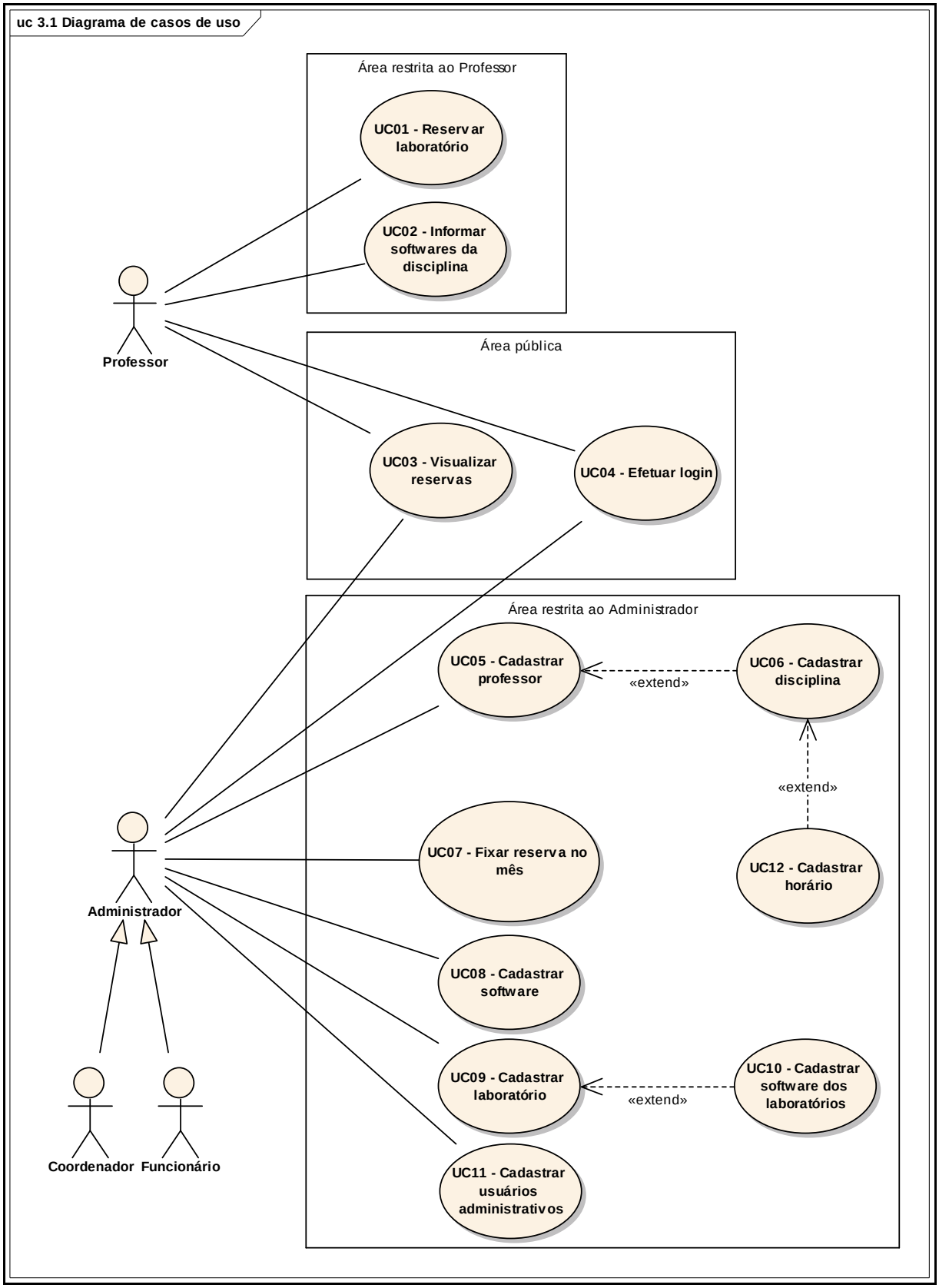


Figura 45 – Diagrama de casos de uso

O primeiro caso de uso (Quadro 11), designado de `Reservar laboratório`, pode também ser considerado o caso de uso principal do sistema de reserva de laboratórios, visto que é durante sua execução que uma reserva é efetuada. Ele possui além do cenário principal,

um cenário alternativo, dando ao usuário maior controle sobre o processo de reserva de laboratório, e quatro cenários de exceção, informando possíveis erros encontrados durante o processo.

O segundo caso de uso (Quadro 12), designado `Informar softwares da disciplina`, informa os softwares necessários para a disciplina. Ele possui um cenário principal e um cenário alternativo.

O terceiro caso de uso (Quadro 13), designado `Visualizar reservas`, apresenta todas as informações das reservas registradas no sistema. Ele possui apenas um cenário principal.

O quarto caso de uso (Quadro 14), designado `Efetuar login`, permite que um professor ou administrador possa ter acesso ao sistema. Ele possui um cenário principal e um cenário de exceção, informando possíveis erros encontrados durante o processo.

O quinto caso de uso (Quadro 15), designado `Cadastrar professor`, permite visualizar, incluir, alterar ou excluir professores. Ele possui um cenário principal e três cenários alternativos.

O sexto caso de uso (Quadro 16), designado `Cadastrar disciplina`, permite visualizar, incluir, alterar ou excluir as disciplinas relacionadas aos professores. Ele possui um cenário principal e três cenários alternativos.

O sétimo caso de uso (Quadro 17), designado `Fixar reserva no mês`, permite reservar laboratórios para as disciplinas que são 100% em laboratório. Ele possui um cenário principal e quatro cenários de exceção, informando possíveis erros encontrados durante o processo.

O oitavo caso de uso (Quadro 18), designado `Cadastrar software`, permite visualizar, incluir, alterar ou excluir softwares que estão instalados na universidade. Ele possui um cenário principal e dois cenários alternativos.

O nono caso de uso (Quadro 19), designado `Cadastrar laboratório`, permite visualizar, incluir, alterar ou excluir laboratórios para serem reservados. Ele possui um cenário principal e três cenários alternativos.

O décimo caso de uso (Quadro 20), designado `Cadastrar software dos laboratórios`, informa os softwares instalados nos laboratórios da universidade. Ele possui um cenário principal e um cenário alternativo.

O décimo primeiro caso de uso (Quadro 21), designado `Cadastrar usuários administrativos`, permite visualizar, incluir, alterar ou excluir usuários administrativos. Ele

possui um cenário principal e dois cenários alternativos.

UC01 – Reservar laboratório: através deste caso de uso é possível efetuar reserva de um laboratório para um professor.	
Pré-condição 01	O professor deve estar cadastrado no sistema.
Pré-condição 02	Professor deve ter acesso à área restrita ao professor e estar logado no sistema.
Pré-condição 03	Devem haver laboratórios e softwares previamente cadastrados no sistema.
Pré-condição 04	O professor deve possuir disciplina(s) cadastrada(s) no sistema.
Cenário principal	1. A partir do menu na opção "Reservas->Reservar Laboratório", o professor seleciona seu nome. 2. O sistema apresenta a lista de disciplinas do professor. 3. O professor seleciona uma disciplina. 4. O sistema apresenta a lista contendo o dia da semana e horário das aulas da disciplina. 5. O professor seleciona um dia da semana e horário da disciplina. 6. O sistema verifica quais os laboratórios têm todos os softwares necessários para a disciplina. 7. O sistema apresenta os laboratórios que possuem os softwares necessários para a disciplina. 8. Nesta etapa o professor seleciona qual laboratório deseja reservar. 9. O professor informa a data da reserva normal. 10. O professor seleciona a opção "Reservar". 11. O sistema registra a reserva do laboratório. 12. Volta ao menu.
Fluxo alternativo	No passo 6, ao verificar os laboratórios que possuem todos os softwares necessários para a disciplina, caso o sistema não encontre nenhum laboratório, o professor pode optar por cancelar a reserva. Volta ao passo 1.
Exceção 01	No passo 6, caso não encontre nenhum laboratório disponível com os softwares necessários para a disciplina, o sistema não carrega nenhum laboratório e o professor não consegue efetuar a reserva.
Exceção 02	No passo 10, o sistema apresenta uma mensagem apropriada, caso o laboratório selecionado pelo professor não esteja disponível. O professor pode optar por listar todos os dados das reservas efetuadas através do menu "Relatório de Reservas" e cancelar a reserva.
Exceção 03	No passo 10, o sistema apresenta uma mensagem apropriada, caso a data informada pelo professor não seja válida.
Exceção 04	No passo 10, o sistema apresenta uma mensagem apropriada, caso a data de reserva informada pelo professor seja diferente do dia da semana do horário da disciplina. O professor pode optar por listar todos os dados das reservas efetuadas através do menu "Reservas->Relatório de Reservas" e cancelar a reserva.
Pós-condições	Uma reserva criada no sistema.

Quadro 11 – Caso de uso 01

UC02 – Informar softwares da disciplina: através deste caso de uso é possível informar os softwares necessários para a disciplina.	
Pré-condição 01	O professor deve estar cadastrado no sistema.
Pré-condição 02	O professor deve possuir disciplina(s) cadastrada(s) no sistema.
Cenário principal	1. A partir do menu, o professor informa uma disciplina na opção "Cadastro->Software-Disciplina". 2. O sistema apresenta a lista de todos os softwares. 3. O professor marca quais softwares são necessários para a disciplina e confirma. 4. O sistema registra quais os softwares usados na disciplina. 5. Volta para o menu.
Fluxo alternativo	No passo 3, caso o professor já tenha informado os softwares da disciplina, ao apresentar a lista de softwares o sistema marca os software da disciplina informados anteriormente. Caso o professor deseja retirar o software da disciplina, é só desmarcá-los e “Salvar”.
Pós-condições	Softwares de uma determinada disciplina foram informados.

Quadro 12 – Caso de uso 02

UC03 – Visualizar reservas: através deste caso de uso é possível visualizar todas as informações das reservas efetuadas.	
Cenário principal	1. A partir da tela principal, o usuário escolhe a opção "Relatório de Reservas". 2. O sistema apresenta a Lista de Reservas contendo todas as reservas efetuadas.
Pós-condições	Apresenta uma lista das reservas efetuadas.

Quadro 13 – Caso de uso 03

UC04 – Efetuar login: este caso de uso permite que um professor ou administrador possa ter acesso ao sistema, através do fornecimento do perfil de acesso (Administrador/Professor), usuário/código e senha.	
Pré-condições	Um professor/administrador precisa estar cadastrado no sistema.
Cenário principal	1. O sistema apresenta tela solicitando o perfil de acesso (Administrador/Professor), usuário/código e a senha do operador. 2. O operador preenche os dados (Perfil de acesso, usuário/código e senha) e confirma. 3. O sistema valida o usuário e senha fornecidas. 4. O sistema apresenta a tela principal do sistema de acordo com a área restrita do acesso.
Exceções	Se no passo 3, o usuário ou a senha não puderem ser validados, o sistema apresenta uma mensagem ('Usuário ou senha inválidos') e retorna ao passo 2.
Pós-condições	Um professor/administrador obtém acesso ao sistema.

Quadro 14 – Caso de uso 04

UC05 – Cadastrar professor: através deste caso de uso é possível visualizar, incluir, alterar ou excluir professores para que tenham acesso ao sistema e efetuem reservas.	
Pré-condições	Coordenador deve ter acesso à área administrativa e estar logado no sistema.
Cenário principal	1. Através da opção "Cadastros->Professores", o administrador acessa o cadastro de professores. 2. O sistema apresenta a lista com os professores cadastrados. 3. O administrador seleciona a opção "Inserir". 4. O administrador informa os dados do professor. 5. O administrador seleciona a opção "Salvar" na tela de Inserir/Editar Professor. 6. Os dados do professor são registrados no sistema. Volta ao passo 2.
Fluxo alternativo 01	No passo 2, ao listar os professores cadastrados, o administrador pode optar por alterar os dados do professor selecionado. 2.1. O administrador seleciona a opção "Editar". 2.2. O administrador altera os dados do professor e seleciona a opção "Salvar" na tela de Inserir/Editar Professor. 2.3 O sistema altera as informações do professor. Volta ao passo 2.
Fluxo alternativo 02	No passo 2, ao listar os professores cadastrados, o administrador pode optar por excluir o professor selecionado. 2.1 O administrador seleciona a opção "Excluir". 2.2 O sistema exclui todas as informações do professor, caso ele não possua disciplinas e reservas cadastradas (integridade referencial em nível de banco de dados). Volta ao passo 2.
Fluxo alternativo 03	No passo 2, ao listar os professores cadastrados no sistema, o administrador pode optar por cadastrar as disciplinas do professor selecionando a opção "Editar->Disciplinas-Inserir" disponível para cada professor listado. 2.1. Executa UC06.
Pós-condições	Um professor é inserido, editado ou excluído do sistema.

Quadro 15 – Caso de uso 05

UC06 – Cadastrar disciplina: através deste caso de uso é possível visualizar, incluir, alterar ou excluir as disciplinas dos professores.	
Pré-condição 01	Coordenador deve ter acesso à área administrativa e estar logado no sistema.
Pré-condição 02	Professor deve estar cadastrado no sistema.
Cenário principal	1. O sistema apresenta a lista de todas as disciplinas cadastradas para o professor. 2. O administrador seleciona a opção "Inserir". 3. O administrador informa os dados da disciplina (nome da disciplina) e seleciona a opção "Salvar" da tela de Inserir/Editar Disciplina. 4. O sistema registra a disciplina do professor. Volta ao passo 1.
Fluxo alternativo 01	No passo 1, o administrador pode optar por alterar os dados da disciplina selecionando a opção "Editar". 1.1. O administrador altera os dados das disciplinas e seleciona a opção "Salvar". 1.2. O sistema registra as alterações.
Fluxo alternativo 02	No passo 1, o administrador pode optar por excluir uma disciplina do professor selecionando-a a partir da lista e seleciona a opção "Excluir". Volta ao passo 1.
Fluxo alternativo 03	No passo 1, ao listar as disciplinas cadastradas no sistema, o administrador pode optar por cadastrar os horários da disciplina selecionando a opção "Horários->Inserir" disponível para cada disciplina listada. 1.1. Executa UC12.
Pós-condições	Uma disciplina é inserida, editada ou excluída.

Quadro 16 – Caso de uso 06

UC07 – Fixar reserva no mês: através deste caso de uso é possível efetuar reserva fixa para disciplinas que são 100% em laboratório.	
Pré-condições	Coordenador deve ter acesso à área administrativa e estar logado no sistema.
Cenário principal	1. A partir do menu na opção "Reservas->Reserva Fixa", o administrador seleciona o professor. 2. O sistema apresenta a lista de disciplinas do professor. 3. O administrador seleciona uma disciplina. 4. O sistema apresenta a lista contendo o dia da semana e horário das aulas da disciplina. 5. O administrador seleciona um dia da semana e horário da disciplina. 6. O sistema verifica quais os laboratórios têm todos os softwares necessários para a disciplina. 7. O sistema apresenta os laboratórios que possuem os softwares necessários para a disciplina. 8. Nesta etapa o administrador seleciona qual laboratório deseja reservar. 9. O administrador informa a data da reserva fixa (data final para reservas fixas). 10. O administrador seleciona a opção "Reservar". 11. O sistema registra a reserva fixa do laboratório (data atual até a data da reserva fixa informada nos respectivos dias de semana da disciplina). 12. Volta ao menu.
Exceção 01	No passo 10, o sistema apresenta uma mensagem apropriada, caso o laboratório selecionado pelo administrador não esteja disponível. O administrador pode optar por listar todos os dados das reservas efetuadas através do menu "Reservas->Relatório de Reservas" e cancelar a reserva.
Exceção 02	No passo 6, caso não encontre nenhum laboratório disponível com os softwares necessários para a disciplina, o sistema não carrega nenhum laboratório e o administrador não consegue efetuar a reserva.
Exceção 03	No passo 10, o sistema apresenta uma mensagem apropriada, caso a data informada pelo administrador não seja válida.
Exceção 04	No passo 10, caso o administrador informe uma data de reserva fixa (data final) que não corresponda ao dia de semana da disciplina selecionada e que seja maior que a data atual, o sistema registra a reserva sem informar ao administrador. Porém ao registrar a reserva, mesmo que a data informada não seja equivalente ao dia de semana da disciplina, mas seja maior que a data atual, o sistema efetua a correção inserindo as reservas fixas com datas válidas. Essa correção pode ser observada pelo administrador no menu "Reservas->Relatório de Reservas".
Pós-condições	Uma reserva fixa é criada no sistema.

Quadro 17 – Caso de uso 07

UC08 – Cadastrar software: através deste caso de uso é possível visualizar, incluir, alterar ou excluir softwares que estão instalados nos laboratórios da universidade.	
Pré-condições	Funcionário deve ter acesso à área administrativa e estar logado no sistema.
Cenário principal	1. Através da opção "Cadastros->Softwares", o funcionário acessa o cadastro de softwares. 2. O sistema apresenta a lista com os softwares cadastrados. 3. O funcionário seleciona a opção "Inserir". 4. O funcionário informa os dados do software. 5. O funcionário seleciona a opção "Salvar" na tela de Inserir/Editar Software. 6. Os dados do software são registrados no sistema. Volta ao passo 2.
Fluxo alternativo 01	No passo 2, ao listar os softwares cadastrados, o funcionário pode optar por alterar os dados do software selecionado. 2.1. O funcionário seleciona a opção "Editar". 2.2. O funcionário altera os dados do software e seleciona a opção "Salvar" na tela de Inserir/Editar Software. 2.3 O sistema altera as informações do software. Volta ao passo 2.
Fluxo alternativo 02	No passo 2, ao listar os softwares cadastrados, o funcionário pode optar por excluir o software selecionado. 2.1 O funcionário seleciona a opção "Excluir". 2.2 O sistema exclui todas as informações do software, caso ele não possua reservas, disciplinas e laboratórios cadastrados (integridade referencial em nível de banco de dados). Volta ao passo 2.
Pós-condições	Um software é inserido, editado ou excluído do sistema.

Quadro 18 – Caso de uso 08

UC09 – Cadastrar laboratório: através deste caso de uso é possível visualizar, incluir, alterar ou excluir laboratórios para serem reservados.	
Pré-condições	Funcionário deve ter acesso à área administrativa e estar logado no sistema.
Cenário principal	1. Através da opção "Cadastros->Laboratórios", o funcionário acessa o cadastro de laboratórios. 2. O sistema apresenta a lista com os laboratórios cadastrados. 3. O funcionário seleciona a opção "Inserir". 4. O funcionário informa os dados do laboratório. 5. O funcionário seleciona a opção "Salvar" na tela de Inserir/Editar Laboratório. 6. Os dados do laboratório são registrados no sistema. Volta ao passo 2.
Fluxo alternativo 01	No passo 2, ao listar os laboratórios cadastrados, o funcionário pode optar por alterar os dados do laboratório selecionado. 2.1. O funcionário seleciona a opção "Editar". 2.2. O funcionário altera os dados do laboratório e seleciona a opção "Salvar" na tela de Inserir/Editar Laboratório. 2.3 O sistema altera as informações do laboratório. Volta ao passo 2.
Fluxo alternativo 02	No passo 2, ao listar os laboratórios cadastrados, o funcionário pode optar por excluir o laboratório selecionado. 2.1 O funcionário seleciona a opção "Excluir". 2.2 O sistema exclui todas as informações do laboratório, caso ele não possua reservas e softwares cadastrados (integridade referencial em nível de banco de dados). Volta ao passo 2.
Fluxo alternativo 03	No passo 2, ao listar os laboratórios cadastrados no sistema, o administrador pode optar por cadastrar os softwares do laboratório selecionando a opção "Editar->Softwares->Inserir" disponível para cada laboratório listado. 2.1. Executa UC10.
Pós-condições	Um laboratório é inserido, editado ou excluído do sistema.

Quadro 19 – Caso de uso 09

UC10 – Cadastrar software dos laboratórios: através deste caso de uso é possível informar os softwares instalados nos laboratórios da universidade.	
Pré-condição 01	Funcionário deve ter acesso à área administrativa e estar logado no sistema.
Pré-condição 02	Deve haver laboratórios e softwares previamente cadastrados no sistema.
Cenário principal	1. O sistema apresenta a lista de todos os softwares cadastrados para o laboratório. 2. O funcionário seleciona a opção "Inserir". 3. O funcionário informa o software e seleciona a opção "Ok" da tela de Inserir Software para o Laboratório. 4. O sistema registra o software para o laboratório. Volta ao passo 1.
Fluxo alternativo	No passo 1, o funcionário pode optar por excluir um software do laboratório selecionando-o a partir da lista e seleciona a opção "Deletar". Volta ao passo 1.
Pós-condições	Um software é inserido no(s) laboratório(s).

Quadro 20 – Caso de uso 10

UC11 – Cadastrar usuários administrativos: através deste caso de uso é possível visualizar, incluir, alterar ou excluir usuários administrativos do sistema.	
Pré-condições	Coordenador deve ter acesso à área administrativa e estar logado no sistema.
Cenário Principal	1. Através da opção "Cadastros->Usuários Administrativos", o administrador acessa o cadastro de usuários administrativos. 2. O sistema apresenta a lista com os usuários cadastrados. 3. O administrador seleciona a opção "Inserir". 4. O administrador informa os dados do usuário. 5. O administrador seleciona a opção "Salvar" na tela de Inserir/Editar Usuário Administrativo. 6. Os dados do usuário são registrados no sistema. Volta ao passo 2.
Fluxo alternativo 01	No passo 2, ao listar os usuários administrativos cadastrados, o administrador pode optar por alterar os dados do usuário selecionado. 2.1. O administrador seleciona a opção "Editar". 2.2. O administrador altera os dados do usuário e seleciona a opção "Salvar" na tela de Inserir/Editar Usuário Administrativo. 2.3 O sistema altera as informações do usuário. Volta ao passo 2.
Fluxo alternativo 02	No passo 2, ao listar os usuários administrativos cadastrados, o administrador pode optar por excluir o usuário selecionado. 2.1 O administrador seleciona a opção "Excluir". 2.2 O sistema exclui todas as informações do usuário. Volta ao passo 2.
Pós-condições	Um usuário é inserido, editado ou excluído do sistema.

Quadro 21 – Caso de uso 11

O décimo segundo caso de uso (Quadro 22), designado `Cadastrar horário`, permite visualizar, incluir, alterar ou excluir os horários das disciplinas relacionadas aos professores. Ele possui um cenário principal e dois cenários alternativos.

UC12 – Cadastrar horário: através deste caso de uso é possível visualizar, incluir, alterar ou excluir os horários das disciplinas dos professores.	
Pré-condição 01	Coordenador deve ter acesso à área administrativa e estar logado no sistema.
Pré-condição 02	Disciplina deve estar cadastrada no sistema.
Cenário principal	1. O sistema apresenta a lista de todos os horários cadastrados para a disciplina. 2. O administrador seleciona a opção "Inserir". 3. O administrador informa os dados do horário (dia da semana, hora início e hora fim) e seleciona a opção "Salvar" da tela de Inserir/Editar Horário da Disciplina. 4. O sistema registra o horário para a disciplina. Volta ao passo 1.
Fluxo alternativo 01	No passo 1, o administrador pode optar por alterar os dados do horário selecionando a opção "Editar". 1.1. O administrador altera os dados do horário e seleciona a opção "Salvar". 1.2. O sistema registra as alterações.
Fluxo alternativo 02	No passo 1, o administrador pode optar por excluir um horário de uma disciplina selecionando-o a partir da lista e seleciona a opção "Deletar". Volta ao passo 1.
Pós-condições	Um horário é inserido, editado ou excluído.

Quadro 22 – Caso de uso 12

3.3.2.2 Diagrama de classes

O diagrama de classe representa uma visão de como as classes estão estruturadas e relacionadas. Na Figura 46 é apresentado o diagrama de classes (visão lógica) do sistema de reserva de laboratórios.

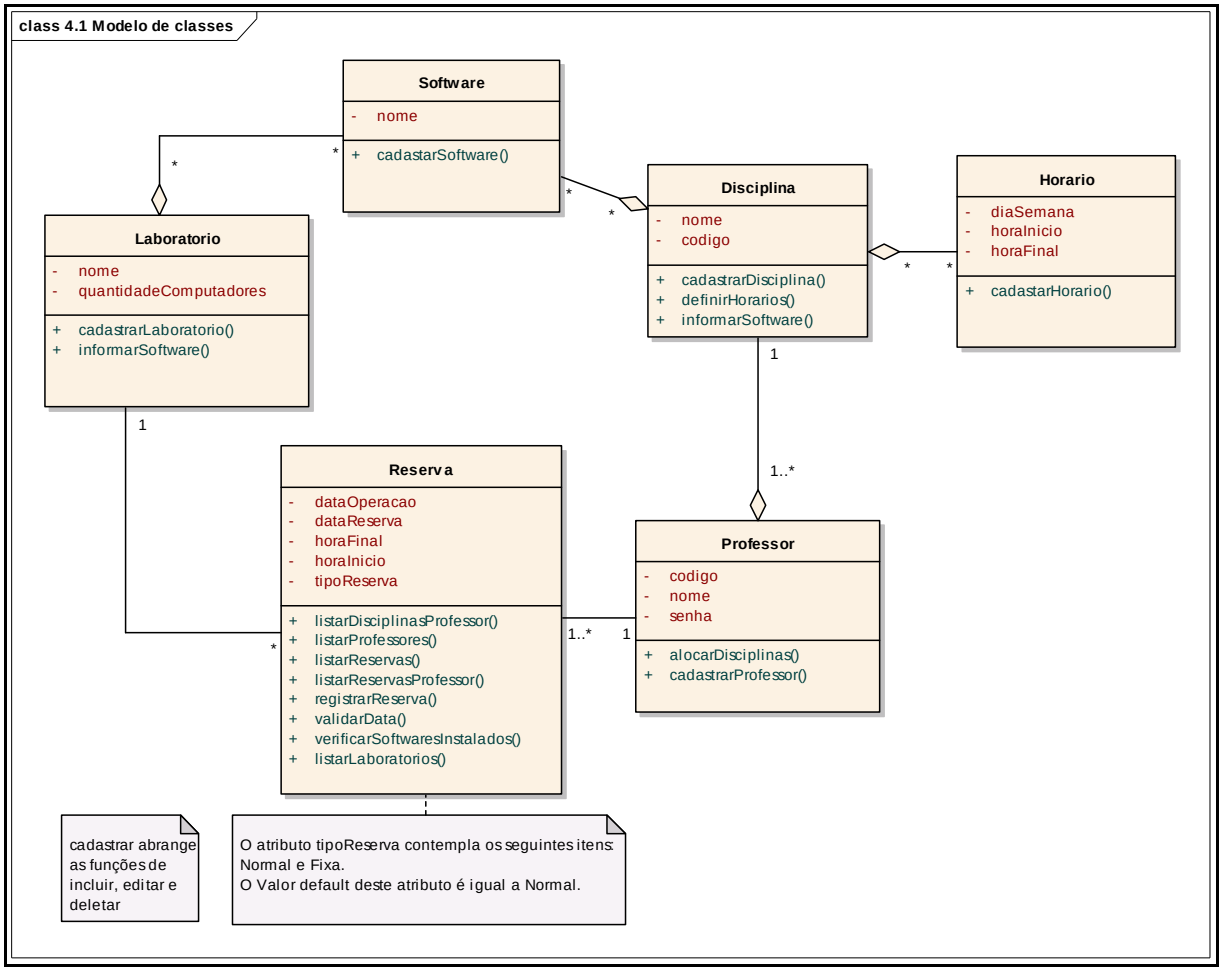


Figura 46 – Diagrama de classes (visão lógica)

A classe principal deste diagrama é `Reserva`, responsável por coletar as informações provenientes de uma reserva de laboratório. Dependendo da área restrita que o usuário possui acesso, a reserva pode ser considerada normal ou fixa. Seus principais métodos são apresentados no Quadro 23.

MÉTODO	DESCRIÇÃO
`registrarReserva`	Realiza o registro de uma reserva normal ou fixa.
`listarProfessores`	Carrega uma lista de professores na reserva fixa, quando o usuário estiver acessando a área administrativa. Lista apenas o professor que está acessando a área restrita ao professor, quando a reserva for normal.
`listarDisciplinasProfessor`	Carrega uma lista de disciplinas relacionadas ao respectivo professor, tanto na reserva fixa, quanto na normal.
`listarLaboratorios`	Carrega uma lista de laboratórios disponíveis de acordo com a disciplina, verificando se os softwares instalados no laboratório estão de acordo com a necessidade da disciplina.

Quadro 23 – Principais funções da classe `Reserva`

3.3.2.3 Diagrama de entidade relacionamento

Para representar o banco de dados MySQL, foi utilizada a ferramenta DBDesigner. A Figura 47 mostra o diagrama de entidade relacionamento físico da base de dados. Este diagrama foi criado a partir do estudo dos requisitos do sistema de reserva de laboratórios.

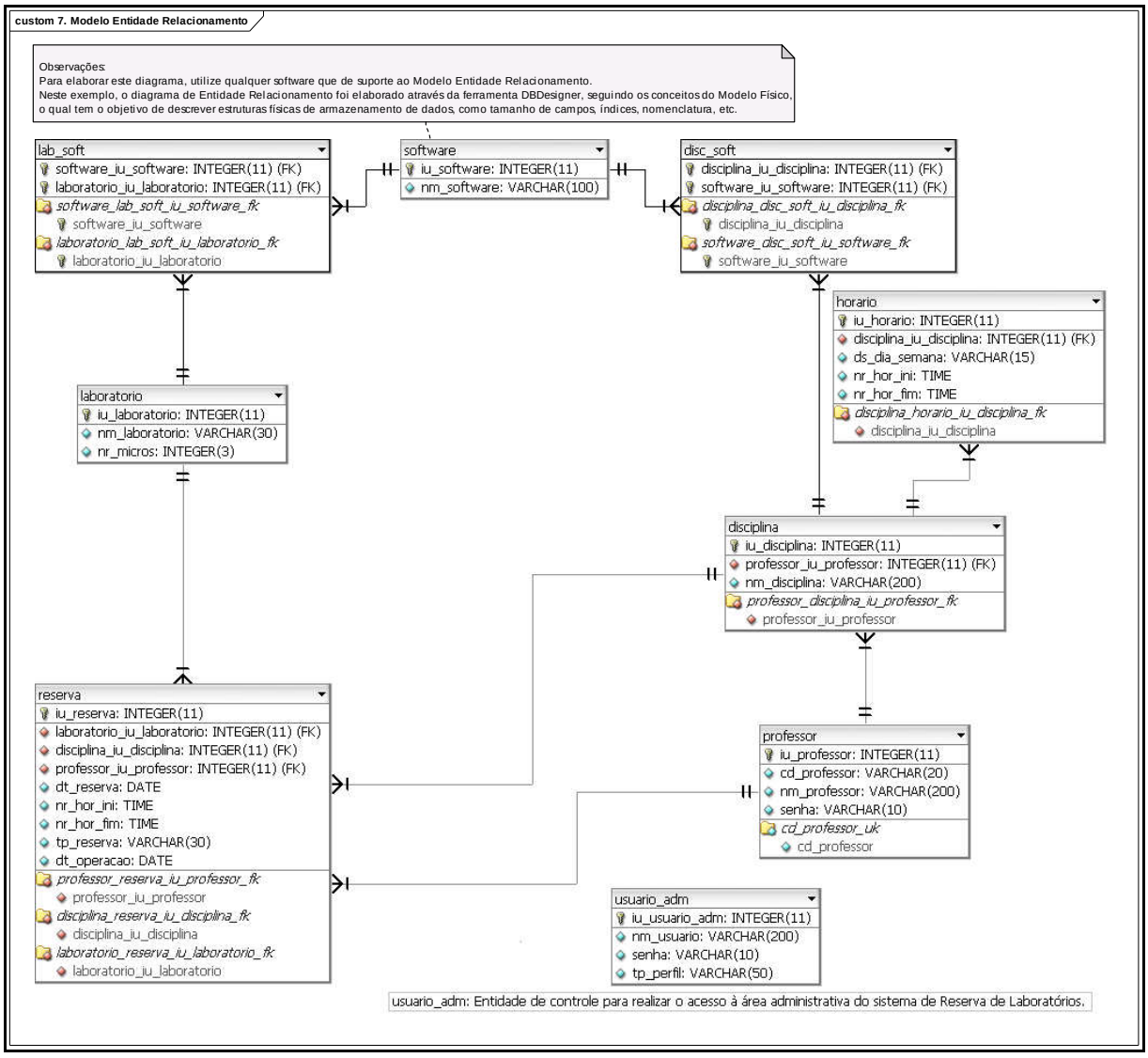


Figura 47 – Diagrama de entidade relacionamento

3.3.3 Implementação

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade do estudo de caso, representado através de um sistema de reserva de laboratórios.

3.3.3.1 Técnicas e ferramentas utilizadas

O sistema de reserva de laboratórios foi implementado utilizando a linguagem de programação Java, através do ambiente NetBeans 5.5 IDE, acessando o banco de dados MySQL versão 4.1. Também optou-se por utilizar o modelo de arquitetura em camadas MVC.

O conceito para o desenvolvimento de sistemas em camadas, utilizando o MVC, foi muito utilizado para o relacionamento organizado entre as classes e para separar a lógica de negócio da apresentação. A tecnologia orientada a objetos está intimamente ligada à arquitetura utilizada para construir o sistema de reserva de laboratórios. Na Figura 48 pode ser observada a organização da arquitetura do sistema, representada através de um diagrama de pacotes.

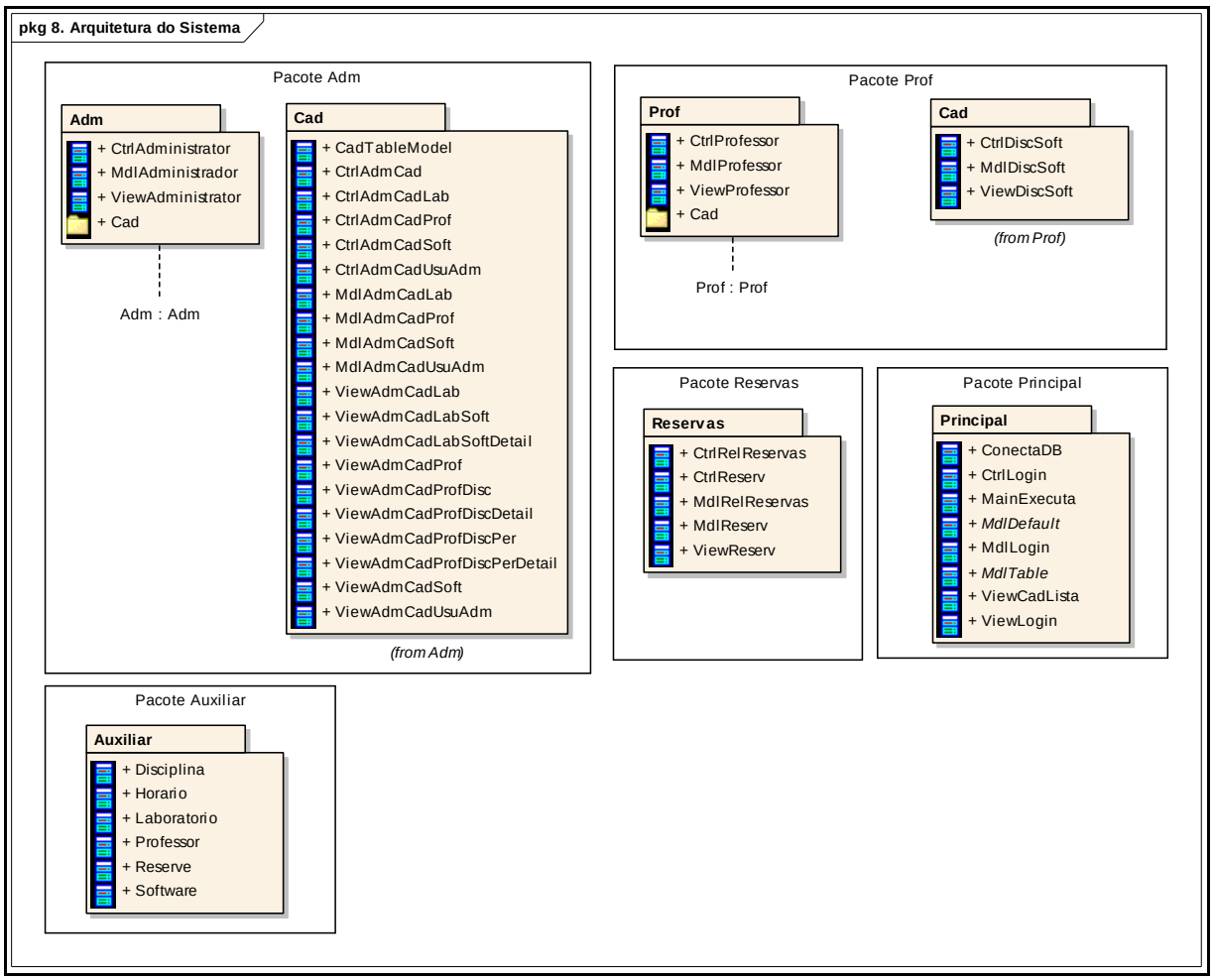


Figura 48 – Diagrama de pacotes representando a arquitetura MVC

Cada pacote (Figura 48) agrega elementos de negócio (`Adm`, `Prof`, `Reservas`, entre outros) com suas respectivas classes. As classes que seguem um prefixo padrão de nomenclatura, iniciando seus nomes com `Mdl`, `View` ou `Ctrl`, identificam em qual camada

MVC cada uma pertence.

Como o objetivo principal deste trabalho é o processo de desenvolvimento de software, serão apresentados apenas os métodos da classe `MdlReserv` (Figura 48), os quais são: `registrarReserva`; `listarProfessores`; `listarDisciplinasProfessor` e `listarLaboratorios`. O código destes métodos foi implementado na camada *Model* do MVC, que representa objetos simples que correspondem aos registros das tabelas no banco de dados.

O método `registrarReserva` (Quadro 24), é responsável por realizar o registro de uma reserva de laboratório do tipo normal ou fixa.

```
public void registrarReserva(int pIProf,Disciplina pDisc,int pILab ,int pIPer
,Date pDt){

  Horario hor = (Horario)arListPer.get(pIPer);
  Laboratorio lab = (Laboratorio)arListLab.get(pILab);
  Professor prof = (Professor)arListProf.get(pIProf);
  if (!verifyDayWeek(pDt,hor.getDiaSemana())){
    JOptionPane.showMessageDialog(null,"O dia informado é diferente do" +
                                       "dia da semana do horário !\n" +
                                       "Dia semana do Horário : " +
                                        hor.getDiaSemana());
  }else{
    Reserva reserv = new Reserva();
    reserv.setIu_professor(prof.getIu());
    reserv.setIu_laboratorio(lab.getIu());
    reserv.setIu_disciplina(pDisc.getIu());
    reserv.setDt_operacao(new Date());
    reserv.setTp_reserva(this.tipo);
    reserv.setDt_reserva(pDt);
    reserv.setNr_hor_ini(hor.getHorIni());
    reserv.setNr_hor_fim(hor.getHorFim());
    reserv.setDiaSemana(hor.getDiaSemana());
      if(!verifyReserved(reserv)){
        ArrayList arReservas = this.getListReserv(reserv);
        for(int i=0 ; i< arReservas.size();i++){
          reserv = (Reserva)arReservas.get(i);
          try {
            st = c.createStatement();
            st.executeUpdate(" INSERT INTO RESERVA "+
                       " (IU_LABORATORIO,IU_DISCIPLINA,IU_PROFESSOR,"+
                         "DT_RESERVA,NR_HOR_INI,NR_HOR_FIM,"+
                         "TP_RESERVA,DT_OPERACAO) "+
                       " VALUES "+
                       " ("+reserv.getIu_laboratorio()+
                       ","+reserv.getIu_disciplina()+
                       ","+reserv.getIu_professor()+
                       ",'" + reserv.getDt_reservaSQL()+
                       "','"+reserv.getNr_hor_ini()+
                       "','"+reserv.getNr_hor_fim()+
                       "','" + this.tipo +
                       "','"+reserv.getDt_operacaoSQL()+"')");
            st.execute("commit");
            st.close();
          } catch (SQLException ex) {
            while (ex != null) {
              System.out.println("SQLState: " + ex.getSQLState());
              System.out.println("Message:  " + ex.getMessage());
              ex = ex.getNextException();
            }
          } catch (java.lang.Exception ex) {
            ex.printStackTrace();
          }
        }
        JOptionPane.showMessageDialog(null,"Reserva realizada"+
                                           " com Sucesso!");
      }else{
        JOptionPane.showMessageDialog(null,"Não foi possível"+
                                           "realizar a reserva" +
                                           "pois existem \n reservas"+
                                           " existentes" +
                                           "no horário, dia"+
                                           " e laboratório");
      }
   }
}
```

Quadro 24 – Código do método `registrarReserva`

O método `listarProfessores` (Quadro 25), é responsável por listar todos os professores durante uma reserva, isso quando a mesma for do tipo fixa e um administrador estiver acessando o sistema. Se a reserva for do tipo normal, apenas o professor que está acessando o sistema aparecerá na lista durante uma reserva.

```
public void listarProfessores(){
  this.arListProf = new ArrayList();
  try {
    String sql = "SELECT IU_PROFESSOR,CD_PROFESSOR,NM_PROFESSOR FROM PROFESSOR";
    if(this.prof.getIu() != 0){
      sql = sql + " WHERE IU_PROFESSOR = " + this.prof.getIu();
    }
    st = c.createStatement();
    rs = st.executeQuery(sql);
    while(rs.next()) {
      this.arListProf.add(new Professor(Integer.parseInt(rs.getString(1)),
                          rs.getString(2),rs.getString(3)));
    }
    rs.close();
    st.close();
  } catch (SQLException ex) {
      while (ex != null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Message:  " + ex.getMessage());
        ex = ex.getNextException();
      }
    } catch (java.lang.Exception ex) {
        ex.printStackTrace();
      }
}
```

Quadro 25 – Código do método `listarProfessores`

O método `listarDisciplinasProfessor` (Quadro 26), é responsável por listar todas as disciplinas de um determinado professor durante uma reserva.

```
public void listarDisciplinasProfessor(){
  this.arListDisc = new ArrayList();
  try {
    st = c.createStatement();
    String sql = "SELECT IU_DISCIPLINA, NM_DISCIPLINA FROM DISCIPLINA";
    if(this.prof.getIu() != 0){
      sql = sql + " WHERE IU_PROFESSOR = " + this.prof.getIu();
    }
    rs = st.executeQuery(sql);
    while(rs.next()) {
      arListDisc.add(new Disciplina(Integer.parseInt(rs.getString(1)),
                     rs.getString(2),this.prof.getCod() ));
    }
    rs.close();
    st.close();
  } catch (SQLException ex) {
      while (ex != null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Message:  " + ex.getMessage());
        ex = ex.getNextException();
      }
    } catch (java.lang.Exception ex) {
        ex.printStackTrace();
      }
    view.refreshDisc();
}
```

Quadro 26 – Código do método `listarDisciplinasProfessor`

O método `listarLaboratorios` (Quadro 27), é responsável por listar todos os laboratórios com os softwares necessários para uma determinada disciplina durante uma reserva.

```
public void listarLaboratorios(Disciplina pDisc){
  this.arListLab = new ArrayList();
  try {
    st = c.createStatement();
    rs = st.executeQuery(" SELECT LAB_SOFT.IU_LABORATORIO," +
                         " LABORATORIO.NM_LABORATORIO " +
                         " FROM lab_soft " +
                         " INNER JOIN LABORATORIO" +
                         " ON (LAB_SOFT.IU_LABORATORIO" +
                         " = LABORATORIO.IU_LABORATORIO) " +
                         " INNER JOIN DISC_SOFT ON" +
                         " (LAB_SOFT.IU_SOFTWARE" +
                         " = DISC_SOFT.IU_SOFTWARE)" +
                         " AND (DISC_SOFT.IU_DISCIPLINA" +
                         " = " + pDisc.getIu() + ")" +
                         " GROUP BY LAB_SOFT.IU_LABORATORIO ," +
                         " LABORATORIO.NM_LABORATORIO" +
                         " HAVING COUNT(LAB_SOFT.IU_SOFTWARE)" +
                         " = (SELECT COUNT(DISC_SOFT.IU_SOFTWARE)" +
                         " FROM DISC_SOFT WHERE IU_DISCIPLINA" +
                         " =" + pDisc.getIu() + ")");
    while(rs.next()) {
      this.arListLab.add(new Laboratorio(Integer.parseInt(rs.getString(1)),
                        rs.getString(2)));
    }
    rs.close();
    st.close();
  } catch (SQLException ex) {
      while (ex != null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Message:  " + ex.getMessage());
        ex = ex.getNextException();
      }
    } catch (java.lang.Exception ex) {
        ex.printStackTrace();
      }
    view.refreshLab();
}
```

Quadro 27 – Código do método `listarLaboratorios`

3.3.3.2 Operacionalidade da implementação

Nesta seção é apresentada a operacionalidade do sistema de reserva de laboratórios. Como o foco do presente trabalho não é a implementação e sim o processo, serão apresentadas apenas as telas principais do sistema.

Inicialmente o usuário deve autenticar-se para acessar sua respectiva área restrita, administrador ou professor (Figura 49).

Figura 49 – Interface de *login*

Logo após o usuário efetuar o acesso ao sistema, sua respectiva área restrita é apresentada (Figura 50).

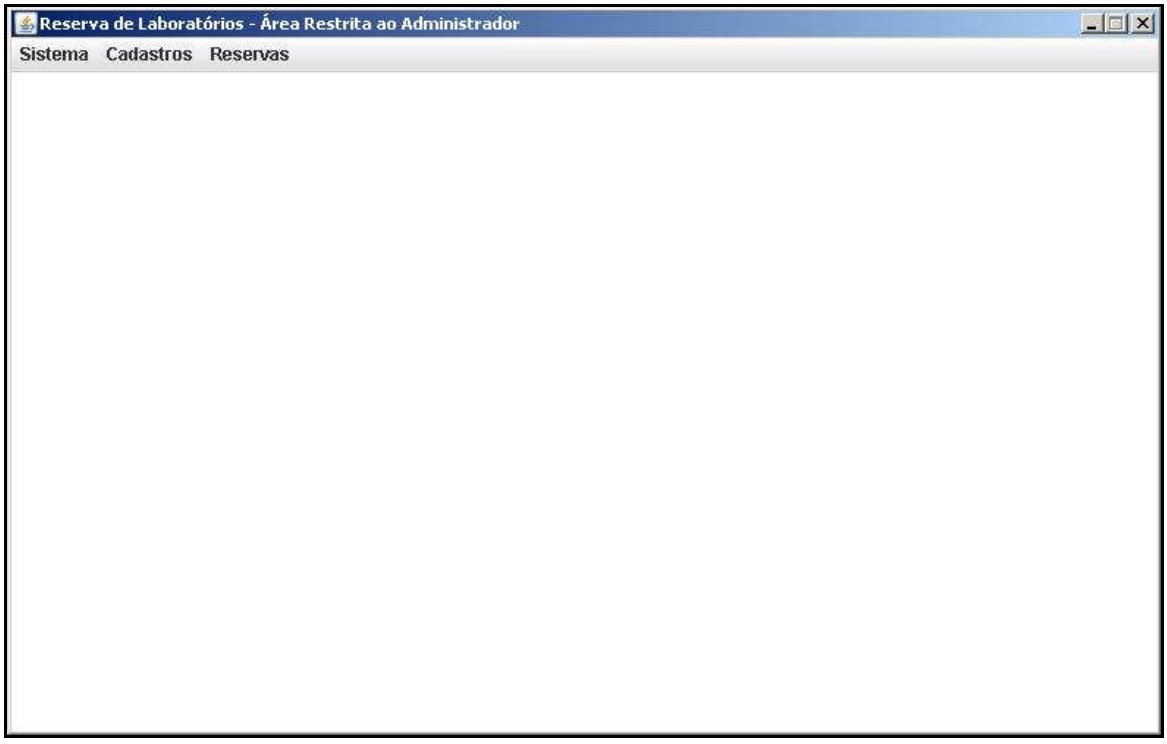


Figura 50 – Interface da área restrita ao administrador

Na opção `Cadastros` (Figura 50), é possível acessar a interface que permite visualizar todos os professores cadastrados no sistema, disponibilizando opções de inserir, editar e excluir um professor (Figura 51).

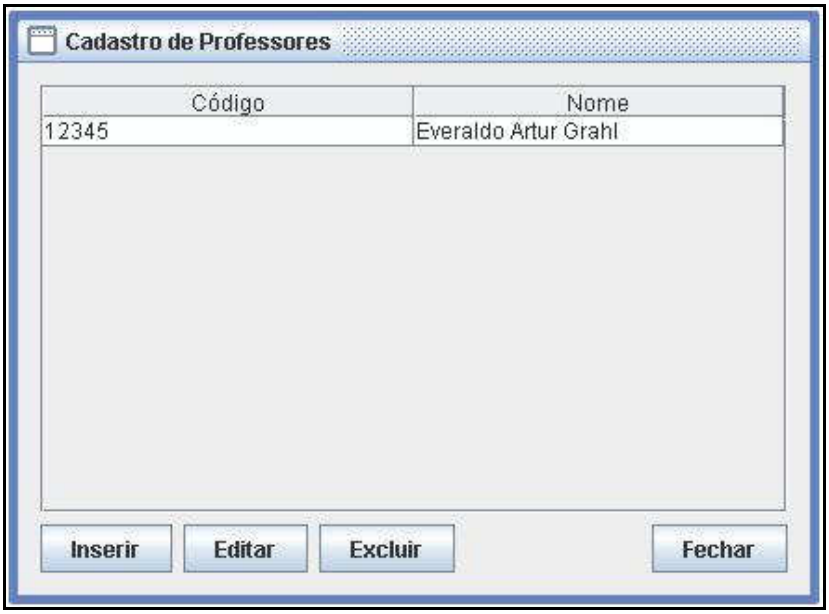


Figura 51 – Interface cadastro de professores

Selecionando a opção `Editar` (Figura 51), é possível, além de editar um professor selecionado, adicionar disciplinas através da opção `Disciplinas` (Figura 52).

Figura 52 – Interface inserir/editar professor

Na opção `Disciplinas` (Figura 52), é possível acessar a interface que permite visualizar todas as disciplinas cadastradas para um professor, disponibilizando opções de inserir, editar e excluir uma disciplina (Figura 53).

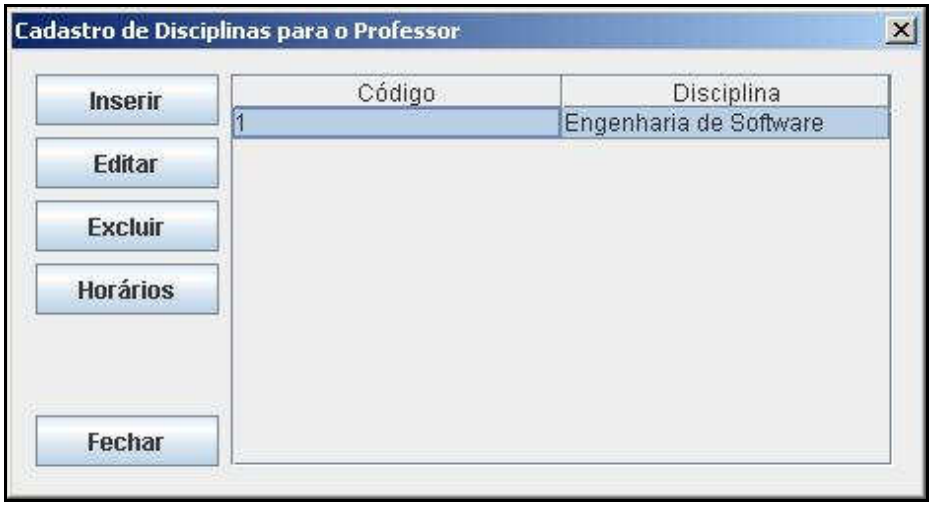


Figura 53 – Interface cadastro de disciplinas para o professor

Na opção `Horários` (Figura 53), é possível acessar a interface que permite visualizar todos os horários cadastrados para uma disciplina, disponibilizando opções de inserir, editar e excluir um horário (Figura 54).

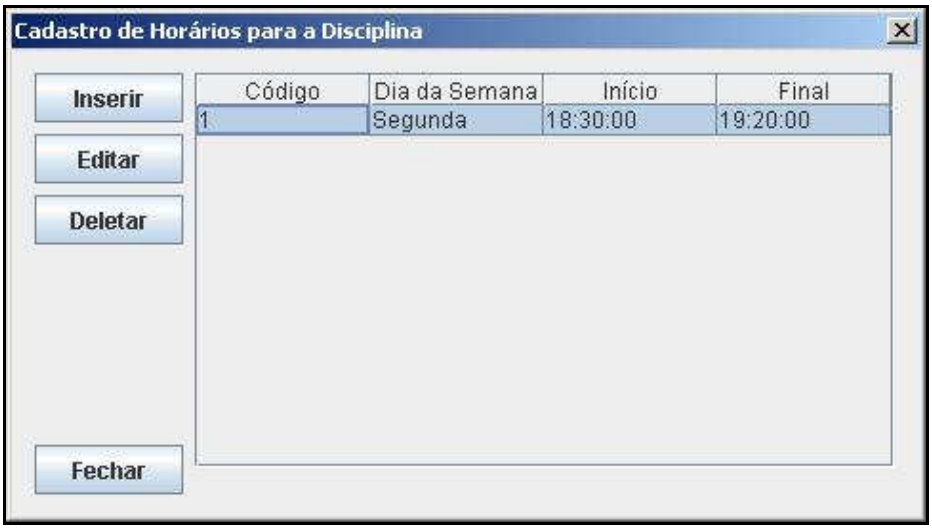


Figura 54 – Interface cadastro de horários para a disciplina

Através da opção `Reservas` (Figura 50), é possível acessar a interface que permite efetuar uma reserva fixa para uma determinada disciplina 100% em laboratório (Figura 55).

Figura 55 – Interface de reservas fixas

Continuando na opção `Reservas` (Figura 50), o usuário pode acessar o relatório de reservas, disponível nas áreas restritas ao administrador e professor. Neste relatório é possível verificar as reservas fixas e normais (Figura 56).

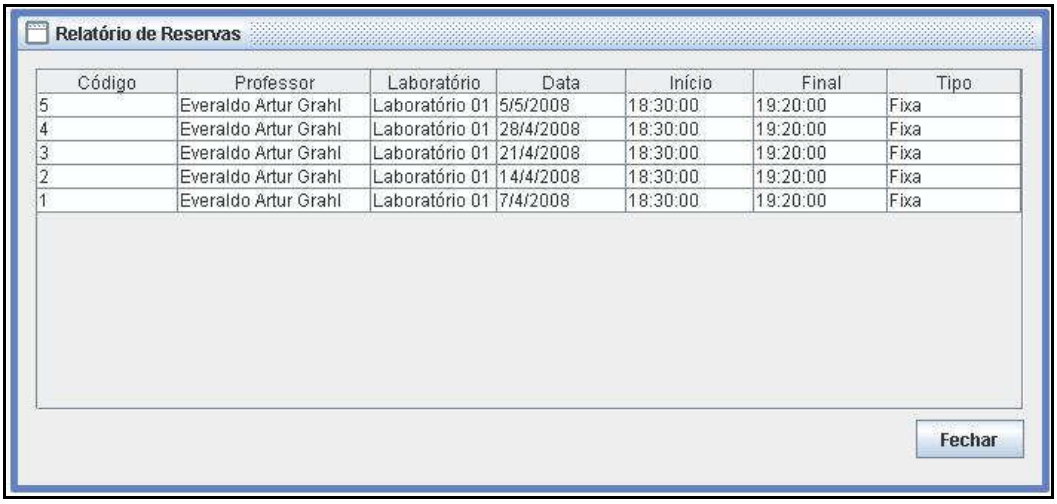


Figura 56 – Interface relatório de reservas

3.4 ELABORAÇÃO DO *TEMPLATE*

Com o objetivo de auxiliar o acadêmico no decorrer das disciplinas da área de ES da FURB, optou-se por disponibilizar um *template* com a especificação completa do estudo de caso apresentado nas seções anteriores. Este *template* agrega os principais artefatos do

FurbUP de uma maneira breve e concisa, servindo de roteiro didático ao acadêmico no desenvolvimento de um projeto de software exigido pelas disciplinas. Foi construído através do EA, utilizando conceitos de orientação a objetos e baseando-se nos diagramas da UML, os quais são: casos de uso; classes; objetos; seqüência; estados e atividades. A estrutura dos conteúdos que compõem o *template* pode ser observada na Figura 57.

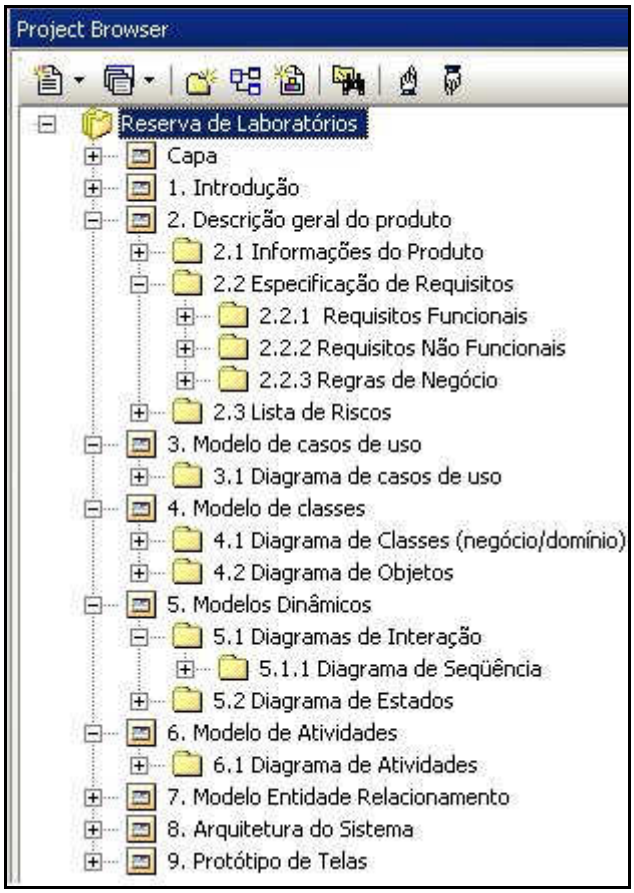


Figura 57 – Estrutura dos conteúdos do *template*

No Quadro 28 podem ser observados quais artefatos estão presentes no *template*, com relação aos produtos de trabalho do FurbUP.

Produtos de Trabalho FurbUP	**Artefatos gerados pelo template**
ANEXO A – Artefato caderno de arquitetura	não cobre
ANEXO B – Artefato lista de itens de trabalho	não cobre
ANEXO C – Artefato lista de riscos	APÊNDICE C – Lista de riscos (*template*)
ANEXO D – Artefato plano de iteração	não cobre
ANEXO E – Artefato plano de projeto	não cobre
ANEXO F – Artefato caso de uso	APÊNDICE D – Caso de uso (*template*)
	Figura 45 – Diagrama de casos de uso
ANEXO G – Artefato especificação de requisitos suplementares	APÊNDICE B – Especificação de requisitos (*template*)
ANEXO H – Artefato visão	APÊNDICE A – Informações do produto (*template*)
ANEXO I – Artefato caso de teste	APÊNDICE F – Caso de teste (*template*)
ANEXO J – Artefato script de teste	
Glossário	APÊNDICE E – Glossário (*template*)

Quadro 28 – Relação entre os artefatos do *template* com os produtos de trabalho FurbUP

O fato do *template* não cobrir alguns produtos de trabalho do processo não afeta a qualidade de sua utilização, pois tanto os conteúdos do *template* como os artefatos FurbUP

estão disponíveis para os professores e alunos. Com isso, ocorre uma flexibilidade na utilização dos conteúdos do processo, deixando por conta dos professores agregarem valor ao projeto de software proposto pelas disciplinas de ES da FURB.

Ao longo de um projeto de software várias evoluções dos artefatos podem ser identificadas, de acordo com a execução das atividades durante as fases do ciclo de vida (apêndice G). O número de versões aumenta de acordo com a quantidade de iterações realizadas em cada fase durante um projeto de software.

3.5 RESULTADOS E DISCUSSÃO

O processo FurbUP possui uma característica vantajosa em relação aos trabalhos correlatos estudados, que é a utilização de um *framework* específico para criação e publicação de processos. Este *framework* é abstraído no ambiente EPFC. O trabalho de Garcia et al. (2004), o processo YP, destina-se ao uso acadêmico, mas não utiliza uma ferramenta para criação de *frameworks* de processos para a sua publicação. Alves e Benitti (2006), através do processo acadêmico PID tem o objetivo de integrar os conteúdos de disciplinas da área de ES como o YP. Também não faz uso de um *framework*, além de não usar uma forma estruturada de publicação. Silva, Souza e Dantas (2006) criaram uma metodologia de desenvolvimento de software para uso acadêmico, através da MD. Sua publicação não é criada através de ambientes de elaboração de processos.

Em relação aos trabalhos correlatos estudados, pode-se afirmar que o FurbUP diferencia-se de todos os trabalhos correlatos por utilizar um *framework* específico para publicação através do ambiente EPFC. No Quadro 29 pode ser observado um comparativo entre o FurbUP e os trabalhos correlatos.

PROCESSO	PROCESSO BASE	USO ACADÊMICO	FORMA DE PUBLICAÇÃO	USO DE *FRAMEWORK* ESPECÍFICO PARA PUBLICAÇÃO
easYProcess	RUP, XP e AM	Sim	Páginas WEB	Não
ProcessID	RUP	Sim	Não consta	Não
MetoDes	RUP, XP e PRÁXIS	Sim	Páginas WEB	Não
FurbUP	OpenUP	Sim	Páginas WEB	Sim

Quadro 29 – Comparativo entre o FurbUP e os trabalhos correlatos estudados

4 CONCLUSÕES

Da análise das necessidades no gerenciamento de um projeto de software em disciplinas de ES da FURB, surgiu a idéia da elaboração do FurbUP, um processo de desenvolvimento de software para uso acadêmico baseado no OpenUP. A elaboração do referido processo voltou-se à utilização no ambiente acadêmico, visando equipes de desenvolvimento pequenas e adjuntas, na produção de um projeto de software no prazo previsto pelas disciplinas.

O EPFC como ferramenta para publicação mostrou-se bastante eficiente e fácil de usar. Uma limitação foi a carência de material de apoio adequado para sua adoção, que foi amenizada a partir de vários testes e abstração dos seus principais componentes que foram comentados ao longo deste texto.

O FurbUP faz uso do método iterativo incremental de desenvolvimento, contendo um ciclo de vida bem definido representado através de fases. Cada uma destas fases possui atividades e tarefas que afetam os produtos de trabalho ao longo das interações, envolvendo constantes modificações nos artefatos gerados pela equipe de desenvolvimento. O FurbUP mostrou-se aderente aos principais objetivos das disciplinas de ES da FURB, por ser compacto, porém completo, e voltado a pequenas equipes de desenvolvimento. Como resultado final do estudo das disciplinas, auxiliado pelo FurbUP, o acadêmico elabora um produto de software com qualidade que pode ser entregue no prazo previsto. Isso faz com que o processo possa ser explorado em sua totalidade pelos acadêmicos no decorrer das disciplinas.

O estudo de caso, representado através de um sistema de reserva de laboratórios, atendeu de forma satisfatória todas as principais tarefas do FurbUP. Os requisitos não funcionais foram atendidos de acordo com a especificação, inclusive o desenvolvimento em camadas utilizando o MVC.

O *template* elaborado no EA, especificado através do estudo de caso desenvolvido, foi essencial no desenvolvimento deste trabalho, por disponibilizar ao acadêmico uma referência que possa auxiliá-lo no desenvolvimento de um projeto de software imposto pelas disciplinas e por aprimorar cada vez mais a utilização de ferramentas CASE.

Os benefícios da utilização de um processo de desenvolvimento de software no ambiente acadêmico, que integra as disciplinas em um único contexto, vêm ao encontro de uma melhora contínua na qualidade de ensino e aprendizagem em ES.

Quanto aos objetivos apresentados, pode-se afirmar que foram alcançados, pois ao final do desenvolvimento deste trabalho chegou-se a um processo de desenvolvimento de software que permite integrar professores e alunos em uma única metodologia na implementação de um produto de software.

Apesar do processo FurbUP ser totalmente baseado no OpenUP, o que caracteriza sua adaptação é a customização de alguns artefatos e o uso de um *template* integrado que cobre boa parte das atividades previstas no processo.

Não houve tempo neste trabalho para explorar o processo por completo, porém com a publicação do mesmo e divulgação para alunos e professores da FURB, os ajustes e adaptações necessárias ficam mais fáceis de serem realizadas em função da ferramenta EPFC.

4.1 EXTENSÕES

Como sugestões para possíveis extensões ao trabalho desenvolvido citam-se:

a) avaliar as atividades previstas durante o desenvolvimento de um projeto de software completo, melhorando os artefatos citados e o *template* disponibilizado;
b) desenvolver interfaces entre o processo criado no ambiente EPFC e as ferramentas CASE mais utilizadas pela FURB, permitindo mais agilidade na construção de um projeto de software;
c) criar outros processos aplicando métodos ágeis, visando trabalhos de desenvolvimento de menor porte.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, Adriana Gomes.; BENITTI, Fabiane Barreto Vavassori. Processo de desenvolvimento integrando disciplinas de engenharia de software. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (SBC) – WORKSHOP DE ENSINO DE INFORMÁTICA (WEI), 26., 2006, Campo Grande. **Anais...** Campo Grande: SBC, 2006. p. 206-215. Disponível em: <http://natalnet.dca.ufrn.br/sbc2006/pdf/arq0050.pdf>. Acesso em: 25 ago. 2007.

AMBLER, Scott W. **Modelagem ágil**: práticas eficazes para a programação extrema e o processo unificado. Tradução Acauan Fernandes. Porto Alegre: Bookman, 2003. 351 p.

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. **Extreme programming**: guia prático. Tradução Kátia Roque. Rio de Janeiro: Campus, 2002. 342 p.

BALDUINO, Ricardo. **Introduction to OpenUP**. [S.l.]. 2007. Disponível em: <http://www.eclipse.org/epf/general/OpenUP.pdf>. Acesso em: 28 mar. 2008.

BEZZERA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002. 286 p.

ECLIPSE. **Eclipse process framework composer**: help. Version 1.2.0.1. [S.l.], 2007a. Documento eletrônico disponibilizado com o Ambiente Eclipse Process Framework Composer.

_____. **OpenUP**. [S.l.], [2007b?]. Disponível em: <http://epf.eclipse.org/wikis/openup/>. Acesso em: 9 maio 2008.

FURB – CIÊNCIAS DA COMPUTAÇÃO. **Disciplina**: engenharia de software. Blumenau, [2008a?]. Disponível em: <http://www.inf.furb.br/bcc/interno.php?secao=580&cd=104>. Acesso em: 08 abr. 2008.

_____. **Disciplina**: processo de software I. Blumenau, [2008b?]. Disponível em: <http://www.inf.furb.br/bcc/interno.php?secao=580&cd=110>. Acesso em: 08 abr. 2008.

_____. **Disciplina**: processo de software II. Blumenau, [2008c?]. Disponível em: <http://www.inf.furb.br/bcc/interno.php?secao=580&cd=115>. Acesso em: 08 abr. 2008.

FURB – SISTEMAS DE INFORMAÇÃO. **Disciplina**: engenharia de software I. Blumenau, [2008a?]. Disponível em: <http://www.inf.furb.br/bsi/interno.php?secao=602&cd=140>. Acesso em: 08 abr. 2008.

_____. **Disciplina**: projeto de software I. Blumenau, [2008b?]. Disponível em: <http://www.inf.furb.br/bsi/interno.php?secao=602&cd=148>. Acesso em: 08 abr. 2008.

_____. **Disciplina**: projeto de software II. Blumenau, [2008c?]. Disponível em: <http://www.inf.furb.br/bsi/interno.php?secao=602&cd=136>. Acesso em: 08 abr. 2008.

GARCIA, Francilene Procópio et al. Easyprocess: um processo de desenvolvimento para uso no ambiente acadêmico. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (SBC) – WORKSHOP DE EDUCAÇÃO EM INFORMÁTICA, 24., 2004, Salvador. **Anais...** Salvador: SBC, 2004. Não paginado. Disponível em: <http://www.dsc.ufcg.edu.br/~pet/Artigos/ARTIGO_YP.pdf >. Acesso em: 25 ago. 2007.

PAPO, José P. L. **RUP**: planejando projetos iterativos - parte 2. [S.l.], 2007. 42 p. Disponível em: <http://www.erudio.com.br/component/option,com_docman/task,doc_view/gid,38/>. Acesso em: 07 set. 2007.

PAULA FILHO, Wilson de Padua**. Engenharia de software**: fundamentos, métodos e padrões**.** 2. ed. Rio de Janeiro: LTC, 2003. 602 p.

SOARES, Michel dos Santos. Uma experiência de ensino de engenharia de software orientada a trabalhos práticos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (SBC) – WORKSHOP DE ENSINO DE INFORMÁTICA (WEI), 24., 2004, Vitória. **Anais...** Vitória: SBC, 2004. Não paginado. Disponível em: <http://www.sbc.org.br/bibliotecadigital/?module=Public&action=PublicationObject&subject=155&publicationobjectid=10>. Acesso em: 18 set. 2007.

SCOTT, Kendall. **O processo unificado explicado**. Tradução Ana M. de Alencar Price. Porto Alegre: Bookman, 2003.

SILVA, Cristiano C.; SOUZA, Kleyton F.; DANTAS, Samuel D. **Metodes metodologia de desenvolvimento de software**. 2006. 159 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Faculdade Cenecista de Brasília, Ceilândia.

SOMMERVILLE, Ian. **Engenharia de software**. 6. ed. Tradução Maurício de Andrade. São Paulo: Addison Wesley, 2003. 592 p.

APÊNDICE A – Informações do produto (*template*)

Na Figura 58 são apresentadas as informações gerais do produto, no caso o sistema de reserva de laboratórios. Estas informações estão disponíveis no *template* oferecido pelo FurbUP.

class 2.1 Informações do Produto

2.1.1 Visão Geral

Baseado na experiência em diversas universidades, verificou-se que professores usam os laboratórios de informática para ministrar algumas de suas aulas. Mas, para utilizar um laboratório, eles devem solicitar ao coordenador do curso que reserve um laboratório para seu horário.

Ao reservar um laboratório, o coordenador leva em consideração outros fatores além da disponibilidade de um laboratório em determinado horário. É verificado se o laboratório já possui instalados os softwares que serão utilizados pela disciplina. Além disso, os laboratórios com maior número de computadores são destinados preferencialmente às turmas com maior número de alunos.

Há também disciplinas que possuem horário fixos em laboratórios, ou seja, as aulas são totalmente feitas em laboratório durante todo o semestre. Isto significa que determinados laboratórios em determinados horários já estão reservados para o professor. O coordenador também mantém registros destas disciplinas fixas em laboratórios para não reservar estes laboratórios para outros professores.

Quando ocorre conflito, ou seja, um professor deseja reservar um laboratório já reservado por outro profesor, este deve negociar com o professor para disponibilizar o laboratório.Se estes entraram em acordo e o professro ceder à reserva, o coordenador deve ser comunicado para atualizar seus registros.

Muitas universidades utilizam vários meios de controlar as reservas de laboratórios. Uma delas é utilizando uma planilha eletrônica, a qual o coordenador matém todos os registros de reservas efetuadas. A planilha contempla várias informações como o dia da reserva, o horário, nome do professor, disciplina e laboratório reservado. Através desta planilha é realizado todo o controle de reservas de laboratório, o que torna o processo lento, pois os professores precisam contatar o coordenador do curso para reservar um laboratório.

Esta planilha de reservas fica disponível ao bolsista que controla as chaves e acessos dos laboratórios. O bolsista, antes de entregar a chave ao professor, verifica na planilha se este reservou o laboratório, e então libera o laboratório para uso do professor.

O sistema automatizará todo este processo de reserva de laboratório. Fornecerá ao professor o controle das suas reservas e eliminará a presença do coordenador para efetuar a reserva. Agilizará o processo e reduzirá o esforço do coordenador. O professor poderá reservar um laboratório a qualquer hora e o bolsista receberá o registro atualizado das reservas instantaneamente, através do relatório de reservas, disponível ao bolsista e aos professores.

O sistema controla, ao reservar um laboratório, se todos os softwares que o professor precisa para sua disciplina estão instalados, não permitindo que o professor reserve um laboratório que não lhe será útil.

O sistema também provê mecanismos para a realização das reservas fixas, no qual o coordenador do curso informa as disciplinas, respectivos laboratórios e horários das reservas, e a data final das reservas fixas, sendo que o sistema efetua todas as reservas durante o período informado.

O sistema será desenvolvido utilizando a linguagem de programação Java e utilizará um banco de dados MySql. Além disso, o sistema será baseado em arquitetura três camadas.

O sistema deverá ser simples e intuitivo para o professor, com visual estruturado, apresentando os passos que deverá tomar ao reservar um laboratório, como escolher os softwares das suas disciplinas, reservar, consultar conflitos e emitir relatório.

2.1.2 Stakeholder

Os envolvidos que utilizaram o sistema de reservas serão na sua maioria professores com experiência e bom conhecimento em informática e Internet. Mas há alguns professores, embora seja a minoria, que não são familiarizados com os sitemas operacionais e sua utilização.

Todos os professores que poderão utilizar o sistema estão cadastrados no banco de dados da universidade e possuem um código pessoal e senha. Os professores utilizarão estes dados para acessar o sistema.

2.1.3 Usuários

Professores e Alunos Bolsistas.

2.1.4 Benefícios do produto

1. Agilidade no processo de reservar laboratórios. (Essencial)

2. Possibilidade de reservar laboratórios a qualquer hora. (Desejável)

3. Maior agilidade para o bolsista ao receber o registro atualizado das reservas. (Essencial)

4. Diminuição de conflitos entre reservas fixas. (Desejável)

2.1.5 Limitações do produto

O sistema de Reserva de Laboratórios não aborda aspectos como:
- solicitações de reservas on-line através da internet.

Figura 58 – Informações do produto (*template*)

APÊNDICE B – Especificação de requisitos (*template*)

No Quadro 30 são apresentados os requisitos funcionais, não funcionais e as regras de negócio do produto, no caso o sistema de reserva de laboratórios. Estas informações estão disponíveis no *template* oferecido pelo FurbUP.

RF01 - O sistema deve efetuar as reservas de laboratórios para as disciplinas.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC01 - Reserva laboratório <3.1 Diagrama de casos de uso>*

RF02 - O sistema deve permitir que o professor verifique quais os conflitos existentes, ou seja, quais professores já reservaram um laboratório em um determinado horário.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC01 - Reserva laboratório <3.1 Diagrama de casos de uso>*

RF03 - O sistema deve cadastrar os softwares utilizados em cada disciplina de cada professor.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC02 - Informa softwares da disciplina <3.1 Diagrama de casos de uso>*

RF04 - O sistema deve apresentar a lista das reservas efetuadas, apresentando a data, horário e para quem está reservado o laboratório.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC03 - Visualiza reservas <3.1 Diagrama de casos de uso>*

RF05 - O sistema deve cadastrar os professores.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC05 - Cadastra professor <3.1 Diagrama de casos de uso>*

RF06 - O sistema deve cadastrar as disciplinas dos professores, assim como os respectivos horários das aulas.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC12 - Cadastra horário <3.1 Diagrama de casos de uso>*
- Realization link from usecase *UC06 - Cadastra disciplina <3.1 Diagrama de casos de uso>*

RF07 - O sistema deve efetuar o registro de reservas fixas (reservas para disciplinas 100% práticas - em laboratório).
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC07 - Fixa reserva no mês <3.1 Diagrama de casos de uso>*

RF08 - O sistema deve cadastrar os laboratórios.
Type: *«Funcional»* **Requisito**
Connections
- Realization link from usecase *UC09 - Cadastra laboratório <3.1 Diagrama de casos de uso>*

RF09 - O sistema deve cadastrar os softwares instalados em cada laboratório.
Type: *«Funcional»* **Requisito**
Connections

- Realization link from usecase *UC10 - Cadastra software dos laboratórios <3.1 Diagrama de casos de uso>*

RF10 - O sistema deve cadastrar os softwares que são utilizados na universidade.
Type: *«Funcional»* **Requisito**

Connections

- Realization link from usecase *UC08 - Cadastra software <3.1 Diagrama de casos de uso>*

RF11 - O sistema deve cadastrar os usuários administrativos.
Type: *«Funcional»* **Requisito**

Connections

- Realization link from usecase *UC11 - Cadastra usuários administrativos <3.1 Diagrama de casos de uso>*

RNF01 - O sistema deve utilizar senhas de acesso para o controle seguro da aplicação.
Type: *«Segurança»* **Requisito**

Connections

- Realization link from usecase *UC04 - Efetua login <3.1 Diagrama de casos de uso>*

RNF02 - O sistema deve ser desenvolvido na linguagem Java.
Type: *«Implementação»* **Requisito**

RNF03 - O sistema deve ser desenvolvido utilizando o ambiente NetBeans 5.5 IDE, seguindo o modelo de arquitetura em camadas conhecida como Model, View e Controller (MVC).
Type: *«Implementação»* **Requisito**

RNF04 - O sistema deve ser desenvolvido na plataforma Windows XP.
Type: *«Implementação»* **Requisito**

RNF05 - O sistema deve utilizar o banco de dados MySQL na versão 4.1 em diante.
Type: *«Implementação»* **Requisito**

RN01 - Caso um professor não consiga reservar um laboratório por motivos de conflito de horário, o mesmo deve tentar negociar com outros professores que possuem reservas neste horário para que estes liberem o laboratório.
Type: *«Negócio»* **Requisito**

Connections

- Realization link from usecase *UC01 - Reserva laboratório <3.1 Diagrama de casos de uso>*

RN02 - O professor só poderá efetuar reservas em laboratórios que contenham os software exigidos pela sua disciplina.
Type: *«Negócio»* **Requisito**

Connections

- Realization link from usecase *UC01 - Reserva laboratório <3.1 Diagrama de casos de uso>*

RN03 - Para ter acesso ao sistema, o professor deve possuir um código pessoal e uma senha no sistema de Reserva de Laboratórios.
Type: *«Negócio»* **Requisito**

Connections

- Realization link from usecase *UC05 - Cadastra professor <3.1 Diagrama de casos de uso>*

RN04 - Somente professores cadastrados podem ter acesso ao sistema.
Type: *«Negócio»* **Requisito**

Connections

- Realization link from usecase *UC04 - Efetua login <3.1 Diagrama de casos de uso>*

Quadro 30 – Especificação de requisitos (*template*)

APÊNDICE C – Lista de riscos (*template*)

Na Figura 59 são apresentados os itens da lista de riscos do produto, no caso o sistema de reserva de laboratórios. Estas informações estão disponíveis no *template* oferecido pelo FurbUP.

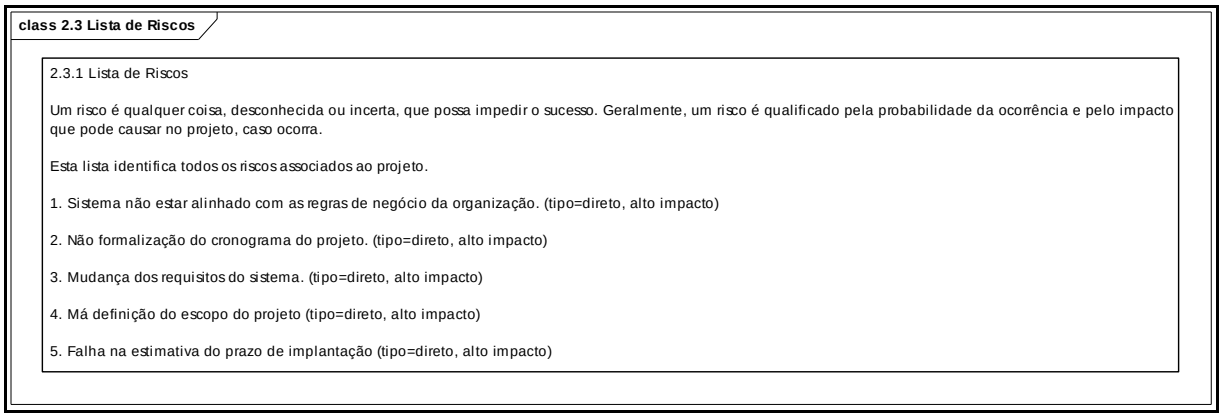


Figura 59 – Lista de riscos (*template*)

APÊNDICE D – Caso de uso (*template*)

No Quadro 31 são apresentados os conteúdos abordados na descrição de um caso de uso. Estas informações estão disponíveis no *template* oferecido pelo FurbUP.

UC01 - Reserva laboratório
Efetuar reserva de um laboratório para um professor.

Linked (System) Requirements

- RF01 - O sistema deve efetuar as reservas de laboratórios para as disciplinas. (*Status:* ; *Dificuldade:* Medium; *Prioridade:* Medium)
- RF02 - O sistema deve permitir que o professor verifique quais os conflitos existentes, ou seja, quais professores já reservaram um laboratório em um determinado horário. (*Status:* ; *Dificuldade:* Medium; *Prioridade:* Medium)
- RN01 - Caso um professor não consiga reservar um laboratório por motivos de conflito de horário, o mesmo deve tentar negociar com outros professores que possuem reservas neste horário para que estes liberem o laboratório. (*Status:* ; *Dificuldade:* Medium; *Prioridade:* Medium)
- RN02 - O professor só poderá efetuar reservas em laboratórios que contenham os software exigidos pela sua disciplina. (*Status:* ; *Dificuldade:* Medium; *Prioridade:* Medium)

Constraints

- *Aprovada Pré-condição* . O professor deve estar cadastrado no sistema.
- *Aprovada Pré-condição* . Professor deve ter acesso à área restrita ao professor e estar logado no sistema.
- *Aprovada Pré-condição* . Deve haver laboratórios e softwares previamente cadastrados no sistema.
- *Aprovada Pré-condição* . O professor deve possuir disciplina(s) cadastrada(s) no sistema.
- *Aprovada Pós-condição* . Uma reserva criada no sistema.

Cenários

Reserva de laboratório {Principal}.
1. A partir do menu na opção "Reservas->Reservar Laboratório", o professor seleciona seu nome.
2. O sistema apresenta a lista de disciplinas do professor.
3. O professor seleciona uma disciplina.
4. O sistema apresenta a lista contendo o dia da semana e horário das aulas da disciplina.
5. O professor seleciona um dia da semana e horário da disciplina.
6. O sistema verifica quais os laboratórios têm todos os softwares necessários para a disciplina.
7. O sistema apresenta os laboratórios que possuem os softwares necessários para a disciplina.
8. Nesta etapa o professor seleciona qual laboratório deseja reservar.
9. O professor informa a data da reserva normal.
10. O professor seleciona a opção "Reservar".
11. O sistema registra a reserva do laboratório.
12. Volta ao menu.

Cancelar reserva {Alternativo}.
No passo 6, ao verificar os laboratórios que possuem todos os softwares necessários para a disciplina, caso o sistema não encontre nenhum laboratório, o professor pode optar por cancelar a reserva. Volta ao passo 1.

Conflito de softwares {Exceção}.
No passo 6, caso não encontre nenhum laboratório disponível com os softwares necessários para a disciplina, o sistema não carrega nenhum laboratório e o professor não consegue efetuar a reserva.

Conflito de disponibilidade {Exceção}.
No passo 10, o sistema apresenta uma mensagem apropriada, caso o laboratório selecionado pelo

professor não esteja disponível.
O professor pode optar por listar todos os dados das reservas efetuadas através do menu "Reservas->Relatório de Reservas" e cancelar a reserva.

Data inválida {Exceção}.
No passo 10, o sistema apresenta uma mensagem apropriada, caso a data informada pelo professor não seja válida.

Dia da semana válido {Exceção}.
No passo 10, o sistema apresenta uma mensagem apropriada, caso a data de reserva informada pelo professor seja diferente do dia da semana do horário da disciplina.
O professor pode optar por listar todos os dados das reservas efetuadas através do menu "Reservas->Relatório de Reservas" e cancelar a reserva.

Quadro 31 – Caso de uso (*template*)

APÊNDICE E – Glossário (*template*)

No Quadro 32 são apresentados os itens do glossário do projeto, no caso o sistema de reserva de laboratórios. Estas informações estão disponíveis no *template* oferecido pelo FurbUP.

Item	Tipo	Significado
Banco de Dados	Técnico	Conjunto de dados persistentes que objetivam atender as necessidades de uma organização.
Conflito	Negócio	Reserva ou tentativa de reserva, de um mesmo laboratório, por dois ou mais professores diferentes, no mesmo horário ou em um horário que toma parte do horário do outro professor.
Coordenador	Negócio	Professor responsável pela administração do curso e que controla todas as reservas solicitadas pelos professores.
Elicitação de Requisitos	Negócio	A elicitação de requisitos é o nome dado às atividades envolvidas com a descoberta dos requisitos (Sommerville & Sawyer, 1997).
Entidade	Técnico	Termo utilizado na abordagem do modelo ER (Entidade relacionamento) no projeto de banco de dados. Representa um conjunto de objetos da realidade modelada.
Ferramenta	Técnico	São softwares de apoio ao desenvolvimento de atividades auxiliando a automatização das informações.
IEEE-STD-830-1998	Negócio	Norma adotada no desenvolvimento do sistema. Responsável pelas regras de especificação e qualidade dos requisitos de software.
Integridade Referencial	Técnico	Termo utilizado na abordagem do modelo ER (Entidade relacionamento) no projeto de banco de dados. A Integridade Referencial é utilizada para garantir a Integridade dos dados entre as entidades relacionadas.
MySQL	Técnico	Banco de dados Open-Source utilizado para o desenvolvimento da estrutura de dados do Sistema de Reserva de Laboratórios.
NetBeans 5.5 IDE	Técnico	Ferramenta de desenvolvimento de Softwares.
Programação	Técnico	Programação (de computadores) é o nome que se dá à atividade de estruturar processos que realizamos mentalmente de forma que possam ser realizados mecanicamente. Um programador transfere para uma máquina (chamada computador) uma tarefa que envolve trabalho mental, da mesma maneira que um engenheiro mecânico transfere para uma máquina (por exemplo, um trator) uma tarefa que envolve trabalho braçal.
Regras de Negócio	Negócio	Regras de Negócio são políticas, condições ou restrições que devem ser consideradas na execução dos processos existentes em uma organização.
Requisito	Técnico	Um requisito é: uma condição ou capacidade necessária para o usuário resolver um problema ou alcançar um objetivo, uma condição ou capacidade que deve ser encontrada ou possuída por um sistema ou componente do sistema para satisfazer um contrato, padrão, especificação ou outro documento imposto formalmente ou uma representação documentada de uma condição ou capacidade.
Requisito Funcional	Técnico	Requisitos funcionais descrevem uma ação que o sistema deve executar, ou seja, identificam os procedimentos que o sistema pode fazer, normalmente em resposta a entrada de dados externa. De maneira bem simples e objetiva, os requisitos funcionais correspondem à listagem de todas as coisas que o sistema deve fazer.

Requisito Não Funcional	Técnico	Requisitos não funcionais são restrições que se colocam sobre como o sistema deve realizar seus requisitos funcionais.
Reserva	Negócio	Exclusividade de uso de um laboratório de informática por um professor em um horário especificado previamente.
Reserva de Laboratórios	Técnico	Sistema de Reserva de Laboratórios a ser desenvolvido conforme especificado neste documento de requisitos.
Software	Técnico	Ferramentas (mecanismos) pelas quais: exploram-se os recursos do hardware, executam-se determinadas tarefas e resolvem-se problemas. Um conceito mais amplo inclui também: instruções que executam uma função desejada, estrutura de dados para manipular informações e documentos para desenvolver, operar e manter programas.
Stakeholders	Negócio	Pessoas envolvidas com o projeto, desde o próprio analista até o usuário do cliente.
Usuários	Negócio	É a pessoa que utiliza o sistema, no caso da Reserva de Laboratórios, os usuários são formados por professores e coordenadores (administrativo).
Windows	Técnico	Plataforma que integra Hardware e Software de uma maneira abstrata para o usuário.

Quadro 32 – Glossário (*template*)

APÊNDICE F – Caso de teste (*template*)

No Quadro 33 são apresentados os conteúdos abordados no caso de teste, exemplificado através de um caso de teste do sistema de reserva de laboratórios. Estas informações estão disponíveis no *template* oferecido pelo FurbUP.

Nome	Caso de Uso	Descrição	Dados de Entrada	Critérios de Aceitação	Última Execução
Reserva de laboratório	UC01 - Reserva laboratório	1. A partir do menu na opção "Reservas->Reservar Laboratório", o professor seleciona seu nome. 2. O sistema apresenta a lista de disciplinas do professor. 3. O professor seleciona uma disciplina. 4. O sistema apresenta a lista contendo o dia da semana e horário das aulas da disciplina. 5. O professor seleciona um dia da semana e horário da disciplina. 6. O sistema verifica quais os laboratórios têm todos os softwares necessários para a disciplina. 7. O sistema apresenta os laboratórios que possuem os softwares necessários para a disciplina. 8. Nesta etapa o professor seleciona qual laboratório deseja reservar. 9. O professor informa a data da reserva normal. 10. O professor seleciona a opção "Reservar". 11. O sistema registra a reserva do laboratório. 12. Volta ao menu.	professor = Everaldo Artur Grahl disciplina = Requisitos de Software dia da semana = Terça hora inicio = 18:30:00 hora fim = 19:20:00 tipo de reserva = Normal data da reserva = 06/05/2008 data da operacao = 05/04/2008	o sistema deve registrar com sucesso a reserva normal	27/5/2008
Data inválida	UC01 - Reserva laboratório	No passo 10, o sistema apresenta uma mensagem apropriada, caso a data informada pelo professor não seja válida.	data da reserva = 2008/31/12	o sistema deve apresentar uma mensagem informando que a data da reserva não é válida	28/5/2008

Nome	Caso de Uso	Descrição	Dados de Entrada	Critérios de Aceitação	Última Execução
Dia da semana válido	UC01 - Reserva laboratório	No passo 10, o sistema apresenta uma mensagem apropriada, caso a data de reserva informada pelo professor seja diferente do dia da semana do horário da disciplina. O professor pode optar por listar todos os dados das reservas efetuadas através do menu "Reservas->Relatório de Reservas" e cancelar a reserva.	data da reserva = 07/05/2008		28/5/2008

Quadro 33 – Caso de teste (*template*)

APÊNDICE G – Evolução dos artefatos durante as iterações

No Quadro 34 são apresentadas as evoluções das versões dos artefatos FurbUP durante as iterações realizadas em cada fase do ciclo de vida. O número de iterações utilizadas é apenas uma simulação para se verificar quantas versões um artefato pode ter durante um projeto de software.

Fases	Atividades	Artefatos Modificados	Iterações Realizadas
Concepção	Iniciar Projeto	Lista de Itens de Trabalho	1
		Visão	
		Glossário	
		Lista de Riscos	
		Plano de Projeto	
	Planejar e Gerenciar a Iteração	Lista de Itens de Trabalho	
		Lista de Riscos	
		Plano de Iteração	
		Plano de Projeto	
	Identificar e Refinar Requisitos	Lista de Itens de Trabalho	
		Glossário	
		Caso de Uso	
		Especificação de Requisitos Suplementares	
		Modelo de Caso de Uso	
		Caso de Teste	
	Concordar na Abordagem Técnica	Caderno de Arquitetura	
Elaboração	Planejar e Gerenciar a Iteração	Lista de Itens de Trabalho	2
		Lista de Riscos	
		Plano de Iteração	
		Plano de Projeto	
	Identificar e Refinar Requisitos	Lista de Itens de Trabalho	
		Glossário	
		Caso de Uso	
		Especificação de Requisitos Suplementares	
		Modelo de Caso de Uso	
		Caso de Teste	
	Desenvolver a Arquitetura	Caderno de Arquitetura	
		Design	
	Desenvolver Incremento de Solução	Design	
		Teste de Desenvolvedor	
		Implementação	
		Caso de Uso	
		Especificação de Requisitos Suplementares	
		Construção	
		Lista de Itens de Trabalho	
		Registro de Teste	
		Descrição do Produto	
		Manual do Usuário	
	Testar a Solução	Script de Teste	

			Lista de Itens de Trabalho	
			Registro de Teste	
			Construção	
		Tarefas Contínuas	Lista de Itens de Trabalho	
	Construção	Planejar e Gerenciar a Iteração	Lista de Itens de Trabalho	2
			Lista de Riscos	
			Plano de Iteração	
			Plano de Projeto	
		Identificar e Refinar Requisitos	Lista de Itens de Trabalho	
			Caso de Uso	
			Especificação de Requisitos Suplementares	
			Modelo de Caso de Uso	
			Caso de Teste	
		Desenvolver Incremento de Solução	Design	
			Teste de Desenvolvedor	
			Implementação	
			Caso de Uso	
			Especificação de Requisitos Suplementares	
			Construção	
			Lista de Itens de Trabalho	
			Registro de Teste	
			Descrição do Produto	
			Manual do Usuário	
		Testar a Solução	Script de Teste	
			Lista de Itens de Trabalho	
			Registro de Teste	
			Construção	
		Tarefas Contínuas	Lista de Itens de Trabalho	
	Transição	Planejar e Gerenciar a Iteração	Lista de Itens de Trabalho	1
			Lista de Riscos	
			Plano de Iteração	
			Plano de Projeto	
		Desenvolver Incremento de Solução	Design	
			Teste de Desenvolvedor	
			Implementação	
			Caso de Uso	
			Especificação de Requisitos Suplementares	
			Construção	
			Lista de Itens de Trabalho	
			Registro de Teste	
			Descrição do Produto	
			Manual do Usuário	
		Testar a Solução	Script de Teste	
			Lista de Itens de Trabalho	
			Registro de Teste	
			Construção	
		Tarefas Contínuas	Lista de Itens de Trabalho	

Quadro 34 – Evolução dos artefatos durante as iterações

ANEXO A – Artefato caderno de arquitetura

No Quadro 35 são apresentados os conteúdos abordados no artefato caderno de arquitetura.

Caderno de arquitetura

1-Propósito / Objetivo

Este documento descreve a filosofia, decisões, limitações, elementos significantes, e qualquer outro aspecto dominante do sistema que modela o projeto/design e implementação.

[Sempre enderece as seções 2 a 6 deste template. Outras seções são recomendadas, dependendo da quantidade de arquitetura desconhecida, da quantidade de manutenção esperada, das habilidades da equipe de desenvolvimento , e da importância de outras preocupações arquiteturais.]

2-Objetivos arquiteturais e filosofia

[Descreve a filosofia da arquitetura. Identifica assuntos que vão levar a filosofia, como: O sistema será conduzido por interesses de distribuição complexa, adaptando-se aos sistemas legados, ou questões de execução? Ele precisa ser robusto para manutenções de longo prazo?

Formula um conjunto de objetivos que a arquitetura necessita satisfazer em sua estrutura e comportamento. Identifica questões críticas que devem ser endereçadas pela arquitetura, como: Há dependências de hardware que devem ser isoladas do resto do sistema? O sistema precisa funcionar eficientemente sob condições incomuns?]

3-Hipóteses e dependências

[Lista as hipóteses e dependências que levam as decisões arquiteturais. Isto pode incluir áreas criticas ou sensíveis, dependências de interfaces de sistemas legados, a habilidade e experiência da equipe, a disponibilidade de recursos, e assim por diante]

4-Requisitos arquiteturalmente significantes

[Insere a referência ou link aos requisitos que devem ser implementados para realizar a arquitetura.]

5-Decisões, limitações e justificativas

[Lista as decisões que devem ser feitas relacionadas ao acesso arquitetural e as limitações sendo colocadas no caminho que o criador cria o sistema. Eles servirão de guias para definir partes do sistema arquiteturalmente significantes. Justifique cada decisão ou limitação para que os criadores entendam a importância da construção do sistema de acordo com o contexto criado por aquelas decisões e limitações. Isso pode incluir um guia de faça ou não faça para guiar criadores na construção do sistema.]

- Decisões ou limitações e justificativas
- Decisões ou limitações e justificativas

6-Mecanismos arquiteturais

[Lista os mecanismos arquiteturais e descreve o estado atual de cada um deles. Inicialmente, cada mecanismo pode ser apenas nome e uma breve descrição. Eles serão desenvolvidos até que o mecanismo seja uma colaboração ou padrão que pode ser aplicado diretamente em algum aspecto do design.]

Mecanismo arquitetural 1

[Descreve o propósito, atributos, e a função do mecanismo arquitetural.]

Mecanismo arquitetural 2

[Descreve o propósito, atributos, e a função do mecanismo arquitetura2.]

7-Abstrações chave

[Lista e descreve brevemente as abstrações chave do sistema. Pode ser uma lista relativamente curta dos conceitos críticos que definem o sistema. As abstrações chave geralmente irão traduzir para classes de analise inicial e padrões importantes.]

8-Camadas ou estrutura arquitetural

[Descreve o padrão arquitetural que você irá utilizar ou como a arquitetura será consistente e uniforme. Isto pode ser uma simples referência para um padrão arquitetural existente ou bem conhecido, como uma Layer Framework, uma referência ao modelo de alto nível da framework, ou uma descrição de como os componentes do sistema principal devem ser postos juntos.]

9-Visões arquiteturais

[Descreve as visões arquiteturais que você utilizará para descrever a arquitetura do software. Ilustra as diferentes perspectivas que você disponibilizará para revisar e documentos de decisões arquiteturais.]

Visões recomendadas

- **Lógica:** Descreve a estrutura e comportamento das porções arquiteturalmente significantes do sistema. Isto pode incluir pacotes de estrutura, interfaces críticas, importantes classes e subsistemas, e as relações entre estes elementos. Pode também incluir visões físicas e lógicas dos dados persistente, se a persistência for usada no sistema. Também é um subsistema do design
- **Operacional:** Descreve os nós físicos do sistema e os processos, processos de execução, e componentes que rodam nestes nós físicos. Esta visão não é necessária se o sistema roda em um único processo e processo de execução.
- **Caso de uso:** Uma lista ou diagrama os casos de uso que contém requisitos arquiteturalmente significantes

Fonte: adaptado de Eclipse (2007b).

Quadro 35 – Artefato caderno de arquitetura

ANEXO B – Artefato lista de itens de trabalho

No Quadro 36 são apresentados os conteúdos abordados no artefato lista de itens de trabalho.

Nome/Descrição	Prioridade	Tamanho estimado (pontos)	Estado	Interação alvo	Atribuido a	Estimativa de tentativas abandonadas (horas)	Horas trabalhadas	Material referência
<<Colunas em azul claro (E-H) são tipicamente usadas quando um item de trabalho é atribuido a uma interação.								
As outras colunas são tipicamente usadas sem considerar se o item de trabalho é atribuido a uma interação ou não.								

Fonte: adaptado de Eclipse (2007b).

Quadro 36 – Artefato lista de itens de trabalho

ANEXO C – Artefato lista de riscos

No Quadro 37 são apresentados os conteúdos abordados no artefato lista de riscos.

<projeto> Lista de Risco									
Identificação do risco	Data identificada	Título	Descrição	Tipo	Impacto	Probabilidade	Magnitude	Proprietário	Estratégia de atenuação
1							0.0		
2							0.0		
3							0.0		
4							0.0		
5							0.0		
6							0.0		
7							0.0		
8							0.0		
9							0.0		
10							0.0		
11							0.0		
12							0.0		
13							0.0		
14							0.0		
15							0.0		
16							0.0		
17							0.0		
18							0.0		

Fonte: adaptado de Eclipse (2007b).

Quadro 37 – Artefato lista de riscos

ANEXO D – Artefato plano de iteração

No Quadro 38 são apresentados os conteúdos abordados no artefato plano de iteração.

Plano de iteração

1-Principais Marcos

[Principais datas mostrando linhas de tempo, como data de inicio e fim; marcos intermediários, pontos de sincronização com outras equipes, demos e assim por diante para a iteração.]

Marco	Data
Início da Iteração	
Término da Iteração	

2-Objetivos de alto-nível

[Lista os objetivos chave para a iteração, tipicamente de um a cinco. Segue exemplos.]

- Endereça assuntos de utilidade levantados pelos Departamentos X.
- Entrega cenários chave que exibem integrações significantes com um Sistema Y.
- Apresenta uma demonstração técnica (demo).

3-Tarefas de item de trabalho

[Esta seção deve referenciar ou a Lista de itens de trabalho, que prove informação sobre quais itens de trabalho devem ser endereçados em qual iteração e por quem, ou especificamente evocar as listas de item de trabalho para endereçar nesta iteração. A solução preferível depende se é ou não trivial para membros do grupo encontrar subdivisões de todos os itens de trabalho que são atribuídas à iteração usando métodos de busca, ao invés de um plano de Iteração.]

Favor veja a lista de itens de trabalho para itens de trabalho a serem endereçados nesta iteração.

4-Assuntos

[Lista qualquer assunto a ser resolvido durante a iteração. Atualiza o status quando novos assuntos são reportados durante as reuniões/encontros diários]

Assunto	Estado	Notas

5-Critério de avaliação

[Uma breve descrição de como avaliar se os objetivos de alto-nível foram satisfeitos. Exemplos a seguir.]

97% dos casos de níveis de sistema aprovados.
Passagem passo a passo da iteração construída pelos departamentos X e Y receberam respostas favoráveis.
Respostas favoráveis para demos técnicas.

6-Avaliação

[Use esta seção para capturar e comunicar resultados e ações de avaliações, que são tipicamente feitas no final de cada iteração. Se você não fizer isto, o grupo principal pode não ser capaz de melhorar o modo como desenvolvem software.]

Avaliação Alvo	Pode ser a iteração completa ou apenas um componente específico
Data da avaliação	
Participantes	
Status do projeto	Por exemplo, expresso como vermelho, amarelo ou verde.

- **Avaliação contra objetivos**
  [Documenta se você endereça os objetivos como especificados no plano de iteração.]
- **Itens de trabalho: Planejados comparados com de fato completados**

[Sumarize se todos os itens planejados para serem endereçados na iteração foram endereçados e quais itens de trabalho foram adiados ou adicionados.]

- **Avaliação contra resultados do teste de avaliação de critérios**
  [Documenta se os critérios de avaliação foram cumpridos no plano de iteração. Isto pode incluir informações como “Demo para o departamento X foi bem recebido, com algumas preocupações levantas sobre a usabilidade” ou “ 495 casos de teste foram automatizados com 98% de média de aprovação. 9 casos de teste atrasaram devido aos itens de trabalho correspondentes terem sido adiados.”
- **Outras preocupações e desvios**
  [Lista outras áreas que foram avaliadas, como a área financeira, ou desvio de programação/agenda, assim como o feedback do stakeholder não capturado em outro lugar.]

Fonte: adaptado de Eclipse (2007b).

Quadro 38 – Artefato plano de iteração

ANEXO E – Artefato plano de projeto

No Quadro 39 são apresentados os conteúdos abordados no artefato plano de projeto.

Plano de projeto

1-Introdução
[descreve brevemente o conteúdo do plano de projeto.]

2-Organização do Projeto
[Introduza a equipe do projeto, membros da equipe, e papéis que representam durante este projeto. Se aplicável, introduza áreas de trabalho domínios, ou outros pacotes de trabalho técnico que são atribuídos a membros da equipe. Introduza projetos próximos, relações e canais de comunicação. Se os projetos são introduzidos em algum outro lugar, referencie a localização com um link.]

Membro do time	Papel A	Papel B
John	X	
Judy		X
Jim	X	X

3-Desenvolvimento do processo e medidas
[Descreva ou referencie quais processos de desenvolvimento serão usados, como FurbUP e liste quaisquer mudanças. Especifique como você irá trilhar o processo, como através de reuniões diárias de revisão, avaliação de iteração, acompanhamento do projeto e acompanhamento dos relatórios de iteração. Discuta como você irá lidar com outras medidas, como o uso do ponto de sistema para estimar o tamanho.]

4-Marcos do projeto e objetivos
[Defina e descreva os objetivos de alto-nível para as fases e defina os marcos. Por exemplo, use a seguinte tabela para esquematizar a programação]

Fase	Iteração	Objetivos primários (riscos e cenários de caso de uso)	Inicio da programação e ou marcos	Velocidade alvo
Concepção	I1	Objetivos 1. Atenuar risco 1 Caso de Uso 1 Cenário 1 Caso de Uso 3 Cenário 2	Data de/Data até	15
Concepção	I2	Objetivos 1. Atenuar risco 2 Caso de Uso 1 Cenário 2	Data de/Data até	16

5-Implantação
[Esquematize a estratégia para implantar o software (e suas atualizações) em um ambiente de produção.]

6-Lições aprendidas
[Liste lições aprendidas da retrospectiva, com ênfase especial em ações tomadas para aprimorar, por exemplo: o ambiente de desenvolvimento, o processo ou a colaboração da equipe.]

Fonte: adaptado de Eclipse (2007b).

Quadro 39 – Artefato plano de projeto

ANEXO F – Artefato caso de uso

No Quadro 40 são apresentados os conteúdos abordados no artefato caso de uso.

Caso de uso: <Nome do caso de uso>

1-Breve descrição

<breve descrição do caso de uso>

2-Breve descrição do ator

2.1-<Nome do Ator 1>

3-Pré-condições

<pré-condição 1>

4-Fluxo básico de eventos

1. O caso de uso inicia quando <ator>, <faz algo>…
2. <Passo 1 do fluxo básico 1>
3. …
4. <Passo n do fluxo básico>
5. O caso de uso termina.

5-Fluxos alternativos

5.1-<Fluxo alternado 1>

Se no passo <x> do fluxo básico o <ator ou sistema faz algo, então

1. <descreva o fluxo>
2. O caso de uso recomeça no passo <y>

6-Subfluxos

6.1<subfluxo 1>

1. <subfluxo 1, passo 1>
2. …
3. <subfluxo 1, passo n>

7-Cenários chave

7.1-<cenário 1>

1. <cenário 1, passo 1>
2. …
3. <cenário 1, passo n>

8-Pós-condições

8.1-<pós-condição 1>

9-Requisitos especiais

<requisito especial 1>

Fonte: adaptado de Eclipse (2007b).

Quadro 40 – Artefato caso de uso

ANEXO G – Artefato especificação de requisitos suplementares

No Quadro 41 são apresentados os conteúdos abordados no artefato especificação de requisitos suplementares.

Especificação de requisitos suplementares

1-Introdução

2-Requisitos amplamente funcionais do sistema

[Relato de especificação de requisitos suplementares, não expressado em casos de uso. Exemplos incluem verificação de autenticação, impressão, relatório.]

3-Qualidades do sistema

[Qualidades representam o URPS no FURPS+ classificação de requerimentos suplementares.]

3.1-Usabilidade

[Descreva requisitos para qualidades como a facilidade de uso, facilidade em aprender, padrões de usabilidade e localização.]

3.2-Confiabilidade

[Confiabilidade inclui o produto e/ou a habilidade do sistema em se manter rodando sob stress ou condições adversas. Especifique requisitos para confiabilidade como níveis de aceitação, e como eles serão medidos ou avaliados. Tópicos sugeridos são disponibilidade, freqüência de gravidade de falhas e capacidade de reabilitação.]

3.3-Desempenho

[as características de desempenho do sistema devem ser esquematizadas nesta seção. Exemplos são respostas em tempo, rendimento, capacidade de tempo de inicialização e encerramento.]

3.4-Apoio

[Esta seção indica qualquer requisito que vá aprimorar o apoio e suportabilidade do sistema sendo construído, incluindo adaptabilidade e aprimoramento, compatibilidade, configurabilidade, escalabilidade e requisitos relacionados à instalação, nível de suporte e manutenção.]

4-Interfaces do sistema

[Requisitos de Interface são parte do + no FURPS+ classificação de requisitos de suporte. Defina as interfaces que devem ser suportadas pela aplicação. Deve conter especificação adequada, protocolos, portas e endereços lógicos e assim por diante, para que o software possa ser desenvolvido e verificado contra requisitos de interface.]

4.1-Interfaces do usuário

[Descreva as interfaces de usuário que serão implementadas pelo software. A intenção desta seção é determinar requisitos referentes à interface. O design da interface pode sobrepor os requisitos do processo de acumulação.]

4.1.1-Ver e sentir

[Forneça a descrição do espírito da interface. Seu cliente pode ter-lhe dado exigências particulares como estilo, cores a serem usadas, grau de interação e assim por diante. Esta seção captura os requisitos para a interface em vez do design da interface.]

4.1.2-Requisitos de navegação e Layout

[Capture os requisitos na área maior da tela e como eles devem ser agrupados juntos.]

4.1.3-Consistência

[Consistência na interface do usuário permite que o usuário preveja o que irá ocorrer. Esta seção determina requisitos no uso dos mecanismos a serem empregados na interface do usuário. Isso aplica ambos no sistema e com outros sistemas e pode ser aplicado em diferentes níveis: controle de navegação, formatos e tamanhos da área da tela, localização para entrar/apresentar dados, terminologia.]

4.1.4Personalização do Usuário e Requisitos de Customização

[Requisitos no conteúdo que devem ser mostrados automaticamente aos usuários ou disponíveis baseados nos atributos do usuário. Algumas vezes os usuários são permitidos customizar o conteúdo mostrado ou personalizar o conteúdo mostrado.]

4.2-Interfaces para sistemas ou dispositivos externos

[Há algum sistema externo com o qual o sistema deva interfacear? Há alguma limitação na natureza da interface entre o sistema e algum sistema externo, como o formato de dados passado entre estes sistemas e algum protocolo particular utilizado? Considere ambas as interfaces providenciadas e requisitadas.]

4.2.1-Interfaces de Software
[Esta seção descreve as interfaces de outros componentes do sistema de software. Estes podem ser componentes comprados, componentes reutilizados de outra aplicação ou componentes sendo desenvolvidos por subsistemas fora da perspectiva desta Especificação de Requisitos de Software (ERS), mas com a qual a aplicação de software deve interagir.]
4.2.2-Interfaces de hardware
[Esta seção define qualquer interface de hardware a serem suportadas pelo software, incluindo estruturas lógicas, endereços físicos, comportamento esperado e assim por diante.]
4.2.3-Interfaces de comunicação
[Descreva quaisquer interfaces de comunicação para outros sistemas ou dispositivos como áreas locais de rede, dispositivos remotos seriais e assim por diante.]

5-Regras de negócios

[Regras de negócio são definições que definem ou limitam algum aspecto do negócio. Regras do negócio geralmente são representadas por regras de produção quando elas devem ser diretamente executadas em um sistema de Tecnologia da Informação (TI): uma regra de produção é uma definição independente da lógica de programação que especifica a execução de uma ou mais ações em caso destas condições serem satisfeitas. Regras de produção definem a operação semântica para o sistema em uma forma independentemente tecnológica. Elas limitam o comportamento expressado em casos de uso de sistema.]
[Organize este documento em classes de regra, um agrupamento de alto nível de regras reais sobre o **conceito de negócio** com um tipo específico de **lógica de processamento**, exemplo: Regras de avaliação de riscos do administrador ou Regras de validação do cliente.]

5.1-<Nome da classe da regra>

5.1.1-<Nome da Regra e identificação (ID)>
[A descrição define a regra. Pode ser feita em uma linguagem natural tipicamente seguida de uma tabela de decisão ou um padrão como: se (lista de condições) então (lista de ações), exemplo:
Se existem pelo menos 3 itens do mesmo tipo no carrinho de compras do cliente e o valor de cada item é maior que R$30,00 então de ao cliente um recibo do qual o valor é 10% do item mais barato.]

6-Limitações do sistema

[Limitações são parte do + no FURPS+ classificação dos requisitos suplementares. Descreva qualquer design; Implementação ou limitações de aplicação no sistema sendo construído que tenham sido mandadas e devem ser aderidas. Exemplos incluem implementação da linguagem de software, uso prescrito de ferramentas de desenvolvimento, componentes de terceira parte ou biblioteca de classes, plataforma de suporte, limites de recursos e requisitos de forma, tamanho ou peso do hardware resultante abrigando o sistema.]

7-Conformidade do sistema

7.1-Requisitos de licença
[Defina qualquer coerção de licença ou outras restrições de uso que serão exibidas no software.]
7.2-Autenticação, direitos autorais e outras notificações
[Esta seção descreve quaisquer negação de autenticação, garantias, notificações de direitos autorais, logomarca,marca registrada, ou conformidade de assuntos da logomarca para o software.]
7.3-Padrões aplicáveis
[Esta seção descreve por referência qualquer padrão aplicável e as seções especificadas de qualquer padrão que aplicado ao sistema sendo descrito. Por exemplo, isto pode incluir padrões legais, de qualidade e regulação, padrões de usabilidade industriais, Interoperabilidade, internacionalização, conformidade do sistema operacional e assim por diante.]

8-Documentação do sistema

[Descreve os requisitos, para documentação on-line do usuário, sistemas de ajuda, ajuda sobre notificações e assim por diante. Prepara expectativas para a documentação para identificar quem será responsável por criá-las.]

Fonte: adaptado de Eclipse (2007b).

Quadro 41 – Artefato especificação de requisitos suplementares

ANEXO H – Artefato visão

No Quadro 42 são apresentados os conteúdos abordados no artefato visão.

Visão

1-Introdução
2-Posição

2.1-Definição do Problema

[Forneça uma definição resumindo o problema sendo resolvido pelo projeto. Os seguintes formatos devem ser usados:]

O problema de	[descreva o problema]
afeta	[os stakeholders afetados pelo problema]
O impacto do mesmo é	[Qual é o impacto deste problema?]
Uma solução bem sucedida seria	[liste alguns benefícios chave de uma solução bem sucedida]

2.2-Definição da Posição do Produto

[Forneça uma definição completa resumindo, no maior nível, a posição única que o produto pretende preencher no mercado. O seguinte formato deve ser usado:]

Para	[Cliente alvo]
Quem	[definição da necessidade ou oportunidade]
O (nome do produto)	É um [categoria do produto]
Que	[definição do benefício chave: que é a razão convincente para comprar]
Diferente de	[alternativa primária de competição]
Nosso produto	[definição da diferenciação primária]

[Uma definição de posição do produto comunica a intenção e a importância de um projeto para todo o pessoal interessado.]

3-Descrição dos Stakeholders

3.1-Sumário do Stakeholder

Nome	Descrição	Responsabilidade
[Nome do tipo de stakeholder.]	[Breve descrição do stakeholder.]	[Resume as responsabilidades chave do stakeholder com relação ao sistema sendo desenvolvido; isto é, os seus interesses como stakeholder. Por exemplo, este stakeholder: assegura que o sistema será passível de manutenção assegura que terá demanda de Mercado para as características do produto monitora o progresso do projeto aprova financiamento e assim por diante]

3.2-Ambiente do usuário

[Detalhe o ambiente de trabalho do usuário alvo. Aqui estão algumas sugestões:
Número de pessoas envolvidas em completar uma tarefa? Isso irá mudar?
Quando tempo dura um ciclo de tarefa? Quanto tempo é gasto em cada atividade? Isto irá mudar?
Qualquer limitação única de ambiente, móvel, em vôo e assim por diante?
Que plataformas de sistema são usadas hoje? Futuras plataformas?
Quais outras aplicações estão em uso? Esta aplicação precisa ser integrada a elas?
É aqui que extrações do modelo de negócios podem ser incluídas para esquematizar a tarefa e os papéis envolvidos e assim por diante.]

4-Visão geral do produto

4.1-Necessidades e características

[Evite design. Mantenha descrições de características em um nível geral. Foque nas capacidades necessárias e por que (não como) deveriam ser implementadas. Capture a propriedade do stakeholder e lançamento planejado de cada característica.]

Necessidade	Prioridade	Características	Lançamento planejado

5-Outros requisitos do produto

[Em um alto nível, liste padrões aplicáveis, hardware, ou requisitos de plataforma; requisitos de desenvolvimento; e requisitos de ambiente.

Defina a faixa de qualidade de performance, robustez, tolerância a erros e características similares que não são capturadas no ajuste de características..

Tome nota de qualquer limitação de design, limitações externas, hipóteses ou outras dependências que se modificadas irão alterar o documento de **visão**. Por exemplo, uma hipótese pode definir que um sistema operacional específico estará disponível para o hardware designado para o software do produto. Se o sistema operacional não estiver disponível, o documento de **visão** precisará mudar.

Defina qualquer requisito de documentação, incluindo manuais do usuário, ajuda on-line, instalação, requisições de rotulação e empacotamento.

Defina a prioridade destes outros requisitos de produtos. Inclua, se útil, atributos como estabilidade, benefícios, esforço e risco.]

Requisitos	Prioridade	Lançamento planejado

Fonte: adaptado de Eclipse (2007b).

Quadro 42 – Artefato visão

ANEXO I – Artefato caso de teste

No Quadro 43 são apresentados os conteúdos abordados no artefato caso de teste.

[O caso de identificação de teste deve ser único. Em adição, o nome de cada caso de teste deve refletir na intenção do caso de teste, idealmente expresso como condição booleana.]

<Identificação do caso de teste> - <Nome do caso de teste>:
Descrição: [Descreve a condição lógica que o teste deve avaliar. Incluindo o resultado esperado.]
Pré-condições: [Liste condições que devem ser verdadeiras antes que o caso de teste possa começar.]
Pós-condições: [Liste condições que devem ser verdadeiras quando o caso de teste termina.]
Dado requerido: [Identifica o tipo de dado requerido para este caso de teste.]

[Repetir conforme necessidade]

Fonte: adaptado de Eclipse (2007b).

Quadro 43 – Artefato caso de teste

ANEXO J – Artefato *script* de teste

No Quadro 44 são apresentados os conteúdos abordados no artefato *script* de teste.

Nome do Teste	<nome do teste>
Caso de uso testado:	<nome do caso de uso>
Descrição do teste:	[Fornece uma descrição concisa do procedimento usado por este script de teste.]
Pré-condições	[Detalhe qualquer pré-condição para execução deste script de teste]
Pós-condições	[Detalhe as pós-condições para execução deste script de teste]
Notas:	
Resultados (aprovado/Falha/Aviso/Incompleto)	

	PASSO DO TESTE	**RESULTADOS ESPERADOS DO TESTE**	**P**	**F**
1.				
2.				
3.				

TABELA DE DADOS DO TESTE					
	1	**2**	**3**	**4**	**5**
[Campo de dados 1]	[Ajuste de dados 1 valor de entrada para campo 1]				
[Campo de dados 2]	[Ajuste de dados 1 valor de entrada para campo 2]				
[Campo de dados 3]	[Ajuste de dados 1 valor de entrada para campo 3]				

Fonte: adaptado de Eclipse (2007b).

Quadro 44 – Artefato *script* de teste

ANEXO L – Listas de tarefas

No Quadro 45 são apresentados os conteúdos abordados nas tarefas que um papel executa no processo FurbUP.

Papéis	Tarefas	Desc. Tarefa	Passos
Analista	Definir a Visão	Definir a visão para o futuro sistema. Descrever o problema e as características baseado nas solicitações dos Stakeholders	**Identifique os Stakeholders**
			Defina o problema a ser resolvido
			Capture termos e definições para o glossário
			Obtenha os requisitos dos stakeholders e outras fontes
			Defina os limites do sistema
			Identifique as restrições do sistema
			Defina as características do sistema
			Revise o Documento de Visão
	Detalhar Requisitos	Esta tarefa descreve como detalhar requisitos para o sistema	**Crie um Diagrama de Casos de Uso**
			Classifique os Casos de Uso e descreva seus cenários
			Atualize os requisitos e glossário
			Revise os requisitos obtidos
	Encontrar e Esboçar Requisitos	Esta tarefa descreve como encontrar e esboçar requisitos para o sistema de modo que o escopo do trabalho possa ser determinado	**Identifique requisitos funcionais, requisitos não-funcionais e regras de negócios**
			Identifique e atualize termos do glossário
			Revise os requisitos obtidos
			Atualize a Lista de Itens de Trabalho
Qualquer Papel	Solicitar Mudança	Capture e registre solicitações de mudança	**Obtenha informações de solicitações de mudança**
			Atualize a Lista de Itens de Trabalho
Arquiteto	Esboçar a Arquitetura	Esboce a arquitetura através da análise dos requisitos arquiteturalmente significantes e da identificação de restrições arquiteturais, decisões e objetivos	**Identifique requisitos com impactos na arquitetura**
			Avalie recursos disponíveis a serem reusados
			Realize provas de conceito
			Capture as principais decisões de arquitetura

	Refinar a Arquitetura	Refine a arquitetura a um nível apropriado de detalhes a fim de auxiliar o desenvolvimento	**Identifique as Classes de projeto e Subsistemas**
			Projete o Banco de dados
			Atualize a arquitetura
			Valide a arquitetura
Desenvolvedor	Projetar a Solução	Identifique os elementos e planeje interações, comportamentos, relações e dados necessários para realizar alguma funcionalidade	**Compreenda os requisitos e arquitetura proposta**
			Identifique os componentes da solução
			Elabore diagramas de seqüencia para os casos de uso críticos
			Planeje a Interface
			Teste a Interface
			Comunique o design
	Implementar Testes de Desenvolvedor	Implemente um ou mais testes que permitam a validação dos componentes individuais de software através da execução	**Defina testes unitários a serem realizados**
			Defina os resultados esperados
			Realize os testes necessários
	Implementar a Solução	Implementar o código fonte para fornecer uma nova funcionalidade ou para reparar algum defeito	**Identifique oportunidades para reuso**
			Transforme o design em implementação
			Escreva o código fonte
			Avalie a implementação
			Comunique decisões significativas
	Integrar e Criar Construção	Esta tarefa descreve como integrar todas as mudanças feitas pelos desenvolvedores no código base e executar os testes mínimos para validar a construção	**Integre os elementos implementados**
			Realize os testes de integração
			Crie uma baseline do código criado e testado
	Documentar o Sistema	Documente o sistema com a criação de uma ajuda (help) e se necessário um manual de usuário	**Planejar a documentação de ajuda do sistema**
			Elaborar a documentação
			Disponibilizar uma descrição do sistema segundo padrões
Gerente de Projeto	Avaliar Resultados	Demonstre o valor do incremento de solução que foi construído durante a iteração e aplique as lições aprendidas para modificar o projeto ou melhorar o processo	**Revise os resultados da iteração**
			Identifique as lições aprendidas
			Encerre o projeto
	Gerenciar a Iteração	Avalie o status do projeto e identifique quaisquer assuntos bloqueadores e oportunidades. Identifique e gerencie exceções, problemas e riscos. Comunique o status do projeto e gerencie as expectativas dos stakeholders	**Capture e comunique o status**
			Trate as exceções e os problemas
			Identifique e gerencie os riscos
			Gerencie os objetivos

	Planejar a Iteração	É uma tarefa colaborativa que planeja o escopo e as responsabilidades para uma única iteração, definindo metas e critérios de avaliação	**Priorize a Lista de Itens de Trabalho**
			Refine o plano do projeto
			Defina os objetivos da iteração
			Confie trabalho à iteração
			Revise os riscos
			Defina os critérios de avaliação
	Planejar o Projeto	É uma tarefa colaborativa que delineia um consenso inicial de como o projeto entregará a visão do produto. O plano de projeto resultante proporciona uma visão geral do projeto em nível de sumário	**Estabeleça uma equipe com papéis bem definidos**
			Determine o tamanho e o escopo do projeto
			Avalie os riscos
			Defina as iterações e suas entregas
Testador	Criar Casos de Teste	Desenvolva casos de teste e dados de teste para os requisitos a serem testados	**Identifique casos de teste relevantes**
			Descreva os casos de teste relevantes
			Identifique necessidades de dados de teste
			Revise os casos de teste
	Implementar Scripts de Teste	Implemente Scripts de Teste para validar a construção da solução. Organize os Scripts de Teste em suítes, e trabalhe em conjunto para garantir profundidade e largura do feedback do teste	**Selecione Casos de Teste para implementar**
			Projete o Script de Teste
			Implemente o Script de Teste executável
			Defina dados de teste específicos
			Verifique a implementação do Script de Teste
	Execute os Testes	Execute os scripts de teste apropriados, análise os resultados, articule as questões e comunique os resultados dos testes à equipe	**Selecione Scripts de Teste**
			Execute Scripts de Teste na construção
			Análise e comunique os resultados dos testes
			Proporcione feedback para a equipe
Stakeholder	NENHUMA TAREFA	NENHUMA TAREFA	NENHUMA TAREFA

Fonte: adaptado de Eclipse (2007b).

Quadro 45 – Listas de tarefas

ANEXO M – Listas de verificação

No Quadro 46 são apresentados os conteúdos abordados nas listas de verificação do processo FurbUP, de acordo com suas respectivas disciplinas.

Disciplina	Lista de Verificação	Itens da lista de Verificação
Arquitetura	Caderno de Arquitetura	**A estrutura global da arquitetura está clara?**
		Os requisitos suplementares foram tratados adequadamente?
		A arquitetura pode ser entregue pela equipe?
		A arquitetura está mostrando uma estabilidade adequada?
		Em geral, a arquitetura parece sensata?
Desenvolvimento	Design	**O design está compreensível?**
		O design é consistente?
		O design é sustentável?
		O design é determinável?
		O design reflete os objetivos arquiteturais do sistema?
		Os elementos do design são modulares?
		O sistema pode ser implementado com as informações do design?
		O design proporciona informações suficientes para testes de desenvolvedor?
		O design descreve o sistema no nível de abstração apropriado?
		O design suporta uma perspectiva grosso modo do sistema?
		Pacotes e Organização
		Visões
		UML
		Modelagem visual não UML
	Implementação	**A implementação está de acordo com a arquitetura e o design?**
		A implementação é testável?
		A implementação está correta?
		A implementação é compreensível?
		Não há redundâncias?
Gerenciamento de Projeto	Lista de Itens de Trabalho	**Os Itens de Trabalho foram descritos sem ambigüidades?**
		Os Itens de Trabalho foram Priorizados?
		Os Itens de trabalho foram avaliados?
		Os Itens de trabalho estão sendo acompanhados?
		Os Itens de Trabalho programados são do tamanho certo?

	Lista de Riscos	**Todos os potenciais riscos ao projeto foram identificados?**
		Todos os riscos foram descritos sem ambigüidades?
		Todos os principais riscos foram avaliados?
		Existem interdependências entre riscos?
	Plano de Iteração	**Você trabalhou no plano com a equipe?**
		Os objetivos da iteração estão claros?
		Os marcos chave para a iteração foram identificados?
		Os membros da equipe se sentem confiantes a respeito das suas atribuições de Itens de Trabalho?
	Plano de Projeto	**O plano foi elaborado com a equipe?**
		A equipe concordou com as práticas de desenvolvimento para o projeto?
		Os marcos do projeto foram definidos?
		A estratégia de implantação foi discutida?
Requisitos	Caso de Uso	**O caso de uso é significativo e sem ambigüidades?**
		A descrição breve descreve claramente a meta primária do caso de uso?
		Os Atores associados e a troca de informações estão claramente definidos?
		As pré-condições foram especificadas?
		Os fluxos básicos e alternativos estão completos, corretos e consistentes?
		As pós-condições foram especificadas?
		Requisitos não funcionais aplicáveis estão capturados?
	Qualidade de Bons Requisitos	**O requisito está correto?**
		O requisito está completo?
		O requisito está claro?
		O requisito é consistente?
		O requisito é verificável?
		O requisito pode ser acompanhado?
		O requisito é praticável?
		O requisito é independente de design?
		O requisito é atômico?
	Requisitos Suplementares	**Os requisitos Funcionais globais que serão implementados na próxima iteração foram capturados e validados?**
		Os requisitos de Usabilidade que serão implementados na próxima iteração foram capturados e validados?
		Os requisitos de Confiabilidade que serão implementados na próxima iteração foram capturados e validados?
		Os requisitos de Desempenho que serão implementados na próxima iteração foram capturados e validados?

		Os requisitos de Suportabilidade que serão implementados na próxima iteração foram capturados e validados?
		Restrições que devam ser consideradas na próxima iteração foram capturadas e validadas?
		Interfaces externas que devam ser consideradas na próxima iteração foram identificadas?
		Regras de Negócios que serão implementados na próxima iteração foram capturadas e validadas?
		Normas aplicáveis e os requisitos Reguladores de Conformidade que devam ser considerados na próxima iteração foram identificados?
	Visão	**Você explorou completamente qual é o problema por detrás do problema?**
		A declaração do problema está formulada corretamente?
		A lista de Stakeholders está correta e completa?
		Todo mundo concorda com a definição dos limites do sistema?
		Você explorou suficientemente as restrições a serem postas no sistema?
		Você cobriu todos os tipos de restrições, incluindo as políticas, econômicas e ambientais?
		Todas as principais características do sistema foram identificadas e definidas?
		As características solucionarão os problemas que estejam identificados?
		As características são consistentes com as restrições que você tenha identificado?
		Alguém que não esteja familiarizado com o projeto consegue, ao ler o documento de Visão, compreender aquilo que você espera que o projeto alcance?
Teste	Script de Teste	**O script de teste possui um nome único que identifique a condição que o script avalia?**
		O script de teste está de acordo com o caso de teste relacionado?
		O script de teste está consagrado pelo uso e sem ambigüidades?
		Todos os dados exigidos estão especificados?
		Cada condição posterior do caso de teste relacionado é avaliada por um passo no script de teste?

Fonte: adaptado de Eclipse (2007b).

Quadro 46 – Listas de verificação