UNIVERSIDADE DO OESTE DE SANTA CATARINA - UNOESC

ALEXANDRO LUIZ HILLESHEIN

REQUIREMENTS PLUGIN E APF PLUGIN: PLUGINS DO REDMINE PARA GERENCIAMENTO DE REQUISITOS DE SOFTWARE E ANÁLISE DE PONTOS POR FUNÇÃO (APF)

Xanxerê

2012

ALEXANDRO LUIZ HILLESHEIN

REQUIREMENTS PLUGIN E APF PLUGIN: PLUGINS DO REDMINE PARA GERENCIAMENTO DE REQUISITOS DE SOFTWARE E ANÁLISE DE PONTOS POR FUNÇÃO (APF)

Trabalho de Conclusão de Curso apresentado ao curso de Ciência da Computação, Área das Ciências Exatas e da Terra, da Universidade do Oeste de Santa Catarina, Campus de Xanxerê. Como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Orientador: André Forchesatto

Xanxerê

2012

RESUMO

Nos meados dos anos 70 começa a “Crise do Software”, quando ainda a Engenharia de Software inexistia. Dentre outros fatores, podem-se citar como principais o não esclarecimento das funcionalidades e características do software, e também a falta de estimativas de prazo, custo e pessoas necessários para o desenvolvimento dos projetos. Estes problemas na área de desenvolvimento de software justificam a Engenharia de Requisitos e a Medição de Software, que possibilitam estimar e gerenciar o mínimo de recursos necessários para o desenvolvimento, atendendo também as expectativas dos clientes. Porém, muitas organizações mesmo cientes do risco de desenvolver um projeto de software, sem um bom detalhamento prévio dos requisitos, e sem estimar esforço, partem diretamente para o desenvolvimento, como tentativa de diminuir custo e tempo. Existem atualmente várias técnicas de medição de software: Mark II, COSMIC – FFP, UCP, APF, etc. Dentre as técnicas existentes, a mais utilizada atualmente tanto para estimar como para medir tamanho de softwares é Analise de Pontos por Função (APF). Neste projeto desenvolvem-se dois plugins para integrar ao gerenciador de projetos Redmine. Um plugin denominado Requirements Plugin, sua função é gerenciar requisitos para projetos de software, o outro denominado APF Plugin, desenvolvido com base na técnica Análise de Pontos por Função (APF), sua função é auxiliar no processo de medição e estimativa tamanho de software. O desenvolvimento destas ferramentas objetiva facilitar o trabalho de gerenciar requisitos de software, assim como de medir e estimar tamanho para projetos de software.

Palavras-chave: Engenharia de software. Requisitos. Medição de Software. Redmine.

ABSTRACT

In the mid of 70´s it is started the “Software Crisis”, when the Software engineering did not existed. Among another factors, it can be citied the non-clarification of the features and functionality of the software, and also the lack of time estimates, cost and people necessary for the development of projects. These problems in the software development justify the Requirements Engineering and Software Measurement, which allow us to estimate and manage the minimum resources needed for development, and also meet customer expectations. However, many organizations even aware of the risk of developing a software project without a good detail of the previous requirements, estimating effort and without depart directly to development, as an attempt to reduce cost and time. There are currently several techniques for measuring software: Mark II, COSMIC - FFP, UCP, APF, etc.. Among existing techniques, the most widely used both to estimate how to measure software size is Function Point Analysis (FPA). In this project it is developed two plugging to integrate with Redmine project manager. The plugging called Plug-in Requirements has the function of managing requirements for software projects, the other called APF Plug-in, developed based on the technique of Function Point Analysis (FPA), whose function is to assist in the process of measuring and estimating software size. The development of these tools aims to ease the job of managing software requirements, as well as to measure and estimate the size of software projects.

Keywords: Software engineering. Requirements. Software Measurement. Redmine.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Um contexto vivenciado atualmente é muitas empresas de software partirem para o desenvolvimento de um produto sem especificar detalhadamente os seus requisitos. Esta opção muitas vezes é adotada como forma de diminuir custo e tempo. Esta forma de desenvolver sistemas pode causar sérios problemas posteriores, principalmente em sistemas mais complexos.

Problemas encontrados em um software depois de instalado têm um custo significativamente maior para serem resolvidos, do que se tivessem sido identificados na fase de planejamento. Por isso, a ideia de economizar na fase de planejamento como forma de diminuir tempo e custo é errônea.

Outro panorama vivenciado atualmente é o frequente atraso na entrega dos produtos pelas empresas de softwares. Em muitos casos, deve-se ao fato de dispensar uma técnica ou método de medição de software.

Os métodos de medição funcional são independentes das tecnologias utilizadas para o desenvolvimento do software, facilitando o processo de estimativa, já que se baseiam nas funcionalidades requeridas pelo usuário. No início de um projeto de software, não é possível saber ao certo sua dimensão, porém, após o levantamento dos seus requisitos a estimativa se torna viável.

Diante destes contextos, este projeto propõe a criação de dois plugins para o Redmine, denominados Requirements Plugin e APF Plugin, os quais objetivam facilitar o processo de gerenciamento de requisitos e medição de tamanho para projetos de software, almejando tornar mais frequente a especificação dos requisitos e a aplicação de medição em projetos de software.

1.1 TEMA

Desenvolvimento de dois plugins para o Redmine, a fim de auxiliar no processo de gerenciar requisitos e no processo de medir ou estimar tamanho para projetos de software.

1.2 PROBLEMA

São dois problemas propostos para este trabalho. Um dos problemas referente a gerenciamento de requisitos, e o outro referente a métricas de software.

Focando primeiramente no primeiro problema proposto, gerenciamento de requisitos, pode-se enfatizar que o processo de levantamento de requisitos, é uma etapa fundamental no desenvolvimento de um software, pois trata de descobrir o que o cliente almeja, criando-se uma visão geral das funcionalidades deste sistema, e as restrições sobre estas funcionalidades.

As ferramentas mais usadas atualmente para descrever os requisitos são os editores de texto, o que deixa muito a desejar. O Redmine é um gerenciador de projetos open-source, muito usado atualmente. Ele permite que novas funcionalidades sejam-lhe adicionadas através de plugins, tornando-se assim uma ferramenta ideal para integrar a funcionalidade de gerenciar os requisitos dos projetos.

Focando agora no segundo problema proposto, métricas de software, supomos que o analista, engenheiro de software ou mesmo o próprio desenvolvedor, juntamente com o cliente realizam o levantamento de todos os requisitos do sistema. Posteriormente, o cliente pergunta: “E então! Quanto isto vai me custar?”. Há duas formas de responder a esta pergunta, a primeira “chutando”, e a segunda estimando através de técnicas e experiências da empresa.

Porém, sabe-se que não é comum a utilização de técnicas para medir e estimar esforços em projetos de software, pois são muitos os fatores que influenciam este processo, fazendo-se muitas vezes necessário um profissional com experiência para que se consigam bons resultados, o que por fim, se refletirá em custos.

Com os requisitos de software estabelecidos, bastando esmiuçar todos os requisitos em suas reais funcionalidades, atribuindo posteriormente um valor a cada função, ao final tem-se um valor total, que diz respeito ao tamanho deste software relativo às suas funcionalidades. Considerando-se o processo de como estas funcionalidades serão desenvolvidas, e as restrições ou características gerais têm-se uma estimação de tamanho para este software.

Após integrar a funcionalidade de gerenciar requisito ao Redmine, ele se torna uma ferramenta propícia para integrar também a funcionalidade de medição e estimativa com APF, que se baseia em funcionalidades.

1.3 OBJETIVOS

1.3.1 **Objetivo Geral**

Desenvolver dois plugins para integrar ao gerenciador de projetos Redmine. Um plugin denominado Requirements Plugin, com a finalidade de gerenciar requisitos para projetos de software, o outro denominado APF Plugin, desenvolvido com base na técnica Análise de Pontos por Função (APF), com a finalidade de auxiliar no processo de medição e estimativa de software.

1.3.2 **Objetivos Específicos**

- Conhecer a linguagem de programação Ruby juntamente com o framework Rails;
- Compreender o processo de criação e gestão de requisitos, em um projeto de sofware;
- Compreender as técnicas de Análise de Pontos por Função (APF).
- Compreender o funcionamento do gerenciador de projetos Redmine;
- Entender como criar e integrar plugins ao Redmine;
- Desenvolver um plugin para o Redmine, que auxilie no processo de gerenciar os requisitos em um projeto de software;

- Desenvolver um plugin para o Redmine, com base na técnica Análise de Pontos por Função (APF), para medição e estimativa de software.

1.4 JUSTIFICATIVA

Segundo uma pesquisa realizada pelo The Standish Group em 1994, apenas 16,2% do total de projetos de software eram terminadas com sucesso, e 52,7% dos projetos eram concluídos com atrasos nos cronogramas e acima dos orçamentos e outros 31,1% eram definitivamente cancelados. Segundo a pesquisa a principal causa dos projetos que foram concluídos com sucesso foi o estabelecimento claro das funcionalidades e características requisitadas pelos usuários. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 48)

Percebe-se assim a importância de esclarecer detalhadamente os requisitos do cliente, especificando as características gerais do software, assim como cada funcionalidade e suas características.

Outro fator importante da área da Engenharia de Software é estimar o esforço necessário para o desenvolvimento do sistema. Define-se como esforço tudo que envolve o processo de desenvolvimento do software, como custo financeiro, pessoas envolvidas e tempo para o desenvolvimento.

Muitos projetos são cancelados durante seu desenvolvimento por falta de recursos, pois os mesmos são limitados em uma empresa, ao mesmo tempo em que o projeto é desenvolvido os recursos tendem a escassez. O que justifica a medição de software, para que esta situação não fique fora de controle, estimando o mínimo de recursos necessários para o desenvolvimento, atendendo ao mesmo tempo as expectativas dos clientes.

Portanto, quanto maior o esclarecimento e o detalhamento das funcionalidades e características em um projeto de software, maior a precisão da estimativa e menor as chances do tempo previamente estimado ser ultrapassado, deixando o cliente satisfeito, pois o software será entregue num tempo e custo previamente estimados.

2 ENGENHARIA DE SOFTWARE

A Engenharia de Software surgiu em meados dos anos 70, para tentar contornar problemas na produção de software. Desde os primórdios até a atualidade a Engenharia de Software vem evoluindo, dando à produção de software um tratamento de engenharia, ou seja, objetivando sistematizar e controlar o processo de produção de software.

A Engenharia de Software abrange o desenvolvimento do software como um todo, porém, a parte de enfoque deste trabalho está no que diz respeito a Requisitos e Medição de Software.

2.1 REQUISITO DE SOFTWARE

"Uma condição, característica ou capacidade, determinada no universo das necessidades do usuário, que deve ser atendida por um sistema na forma de aspectos funcionais ou não funcionais." (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 45)

Requisito de software é tudo o que o usuário solicita em um sistema, todas suas funcionalidades e características. Segundo Wazlawick (2004, p. 38), os requisitos de software podem ser divididos em dois grupos:

a) Os requisitos funcionais, que correspondem à listagem de tudo o que o sistema deve fazer.

b) Os requisitos não funcionais, que são restrições colocadas sobre como o sistema deve realizar seus requisitos funcionais.

E ainda segundo Wazlawick (2004, p. 28), os requisitos funcionais podem ainda ser divididos em dois grupos:

a) Os requisitos funcionais evidentes, que são efetuados com conhecimento do usuário. Esses requisitos corresponderão a eventos do sistema e respostas do sistema, ou seja, as trocas de informações que ocorram pela interface do sistema com o meio exterior.

b) Os requisitos funcionais ocultos, que são efetuados pelo sistema sem o conhecimento explícito do usuário.

Os requisitos não funcionais podem ser obrigatórios ou apenas desejados, isto é, aqueles que devem ser obtidos de qualquer maneira e os que podem ser obtidos caso isso não cause maiores transtornos no processo de desenvolvimento. (WAZLAWICK, 2004, p. 39)

Outra classificação útil para os requisitos não funcionais indicará se são permanentes ou transitórios. O requisito permanente nunca mudará com o tempo (por exemplo a interface feita por meio de janelas), já o requisito transitório poderá sofrer alterações no futuro (por exemplo, a forma de calcular impostos).

Segundo Wazlawick (2004, p. 41), ainda em relação aos requisitos não funcionais, existem aqueles diretamente associados a uma função e outros que são gerais para o sistema, ou também conhecidos como requisitos suplementares ou adicionais. Sendo requisitos não funcionais, associados ou adicionais, eles podem ser divididos em várias categorias. Wazlawick (2004, p. 41), sugere algumas categorias:

a) Usabilidade: Os fatores humanos envolvidos no sistema; O tipo de ajuda que o programa deve prover; As formas de documentação ou manuais disponíveis; Como esses manuais vão ser produzidos; O tipo de informação que eles vão conter; Seria interessante definir esses tópicos na fase de concepção, visto que o contrato com o cliente deveria especificar muitas dessas questões. (WAZLAWICK, 2004, p. 41)

b) Confiabilidade: Tratamento de falhas no sistema; O analista não é obrigado a produzir um sistema totalmente tolerante a falhas, mas deve estabelecer que tipo de falhas o sistema será capaz de gerenciar: falta de energia, falha de comunicação, falha de mídia de gravação etc. Não se deve confundir falha com erro de programação, pois erros de programação são elementos que nenhum software deveria conter. As falhas são situações anormais que podem ocorrer mesmo para um software implementado sem nenhum erro de programação. (WAZLAWICK, 2004, p. 41)

c) Performance: a eficiência e a precisão apresentada pelo sistema. Na fase de concepção não se define como o sistema fará para cumprir o requisito, apenas se diz que de alguma

forma ele terá de ser cumprido no projeto. Cabe ao projetista e ao programador garantir que o requisito seja satisfeito. Se o analista por algum motivo conclui que o requisito não pode ser cumprido, então o requisito passa a ser um risco do sistema e eventualmente necessitará de um estudo ainda mais aprofundado na fase de concepção, para verificar a possibilidade de sua realização. (WAZLAWICK, 2004, p. 41)

d) Configurabilidade: o que pode ser configurado no sistema; o que pode ser configurado pelo usuário sem alterações no código do sistema. (WAZLAWICK, 2004, p. 42)

e) Segurança: definir os tipos de usuários e que funções cada um pode executar. (WAZLAWICK, 2004, p. 42)

f) Implementação: definir a linguagem a ser usada, assim como bibliotecas disponíveis, e bancos de dados acessíveis. (WAZLAWICK, 2004, p. 42)

g) Interface: como vai ser a interface? Vai ser seguida alguma norma ergonômica? (WAZLAWICK, 2004, p. 42)

h) Empacotamento: a forma em que o software deve ser entregue ao usuário final. (WAZLAWICK, 2004, p. 42)

i) Legais: muitas vezes uma equipe de desenvolvimento deve contar com uma assessoria jurídica para saber se está infringindo direitos autorais ou normas específicas da área para a qual a qual o software está sendo desenvolvido. (WAZLAWICK, 2004, p. 42)

A tabela 1 abaixo é um comparativo de requisito funcional, com requisitos não funcionais associados.

Tabela 1: Exemplo de requisito funcional, com requisitos não-funcionais associados

F1 Registrar empréstimo				**Oculto( )**
Descrição: O sistema deve registrar empréstimos de fitas, indicando o cliente e as fitas que foram emprestadas, bem como a data do empréstimo e o valor previsto para pagamento na devolução.				
Requisitos Não-Funcionais				
Nome:	**Restrição:**	**Categoria**	**Desejável**	**Permanente**
NF1.1 Controle de Acesso	**A função só pode ser acessada por um usuário com perfil de operador ou superior.**	**Segurança**	**( )**	**(X)**
NF 1.2 Identificação de fitas	**As fitas devem ser identificadas por um código de barras.**	**Interface**	**( )**	**(X)**
NF1.3 Identificação do cliente	**O cliente deverá ser identificado a partir do seu nome.**	**Interface**	**( )**	**( )**
NF1.4 Tempo de Registro	**O tempo de registro de cada fita deve ser inferior a um segundo.**	**Performance**	**(X)**	**( )**
...	**...**			

Fonte: WAZLAWICK (2004, p.).

A tabela 2, abaixo, demonstra exemplos de requisitos suplementares.

Tabela 2: Exemplos de requisitos suplementares.

Nome	Restrição	Categoria	Desejável	Permanente
S1 Tipo de Interface	**As interfaces do sistema devem ser implementadas como formulários acessíveis em um browser HTML.**	**Interface**	**( )**	**( )**
S2 Armazenamento de dados	**A camada de persistência deve ser implementada de forma que diferentes tecnologias de bancos de dados possam vir a ser utilizadas no futuro.**	**Persistência**	**( )**	**(X)**
...	**...**			

Fonte: WAZLAWICK, 2004

Para entendermos a importância dos requisitos de software, recorremos ao seguinte trecho de (VAZQUEZ, SIMÕES E ALBERT, 2005, p. 46)

> Para garantir o sucesso na conclusão de um sistema de software, atendendo às expectativas dos usuários, é fundamental a existência de um gerenciamento eficiente de requisitos capaz de manter o mapeamento das necessidades dos usuários e das características desejáveis para o software, e de identificar as mudanças nos requisitos o mais cedo possível, durante o ciclo de desenvolvimento, evitando que os riscos existentes se materializem e causem problemas.

2.2 MEDIÇÃO DE SOFTWARE

Cada vez mais se busca softwares de maior qualidade, e que se adere a custos e prazos, estabelecidos pelos clientes. Embora tenha se avançado muito nos últimos anos, produzir softwares de baixo custo, boa qualidade e dentro de prazos planejados, ainda é um desafio às organizações de softwares. Tentando melhorar seus processos, as organizações têm buscado alternativas, e uma delas é a medição de software.

A medição de software possibilita planejar e monitorar o processo de produção de software. A partir da medição é possível estimar a quantidade de esforço, ou seja, tempo, pessoas e recursos para a realização do projeto. E a partir do tempo, formar uma baseline para estimativas, ou seja, usar a experiência de projetos passados, para estimar os novos projetos. Além de ser base para o controle de qualidade.

Existem, atualmente, várias técnicas para medição de projetos de software, porém a de maior aceitação é a contagem de pontos de função do Internacional Function Point Users Group (IFPUG), a qual se baseia no ponto de vista do usuário, ou seja, a contagem ocorre baseando-se nas funcionalidades exigidas pelo usuário, propositalmente independente das técnicas utilizadas para o desenvolvimento.

> Empiricamente as principais técnicas de estimativa de projetos de desenvolvimento de um software assumem que o tamanho de um software é um vetor importante para a determinação do esforço para a sua construção. Logo, saber o seu tamanho é um dos primeiros passos do processo de estimativa de esforço, prazo e custo. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 13)

2.2.1 **Análise de Pontos de Função (APF)**

APF é uma técnica padrão para medir sistemas do ponto de vista de seus usuários pela quantificação das funcionalidades fornecidas por um software, tendo por objetivo tornar a medição independente da tecnologia utilizada para a construção do software. Ou seja, a APF busca medir o que o software faz, e não como ele foi construído. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 13).

Segundo Vazquez, Simões e Albert (2005, p. 52), o IFPUG define como objetivos primários da análise de pontos de função:

- Medir a funcionalidade que o usuário solicita e recebe.
- Medir o desenvolvimento e manutenção de software de forma independente da tecnologia utilizada para sua implementação.

E ainda segundo Vazquez, Simões e Albert (2005, p. 52), além dos objetivos o processo de contagem de pontos de função deve ser:

- Simples o suficiente para minimizar o trabalho adicional envolvido no processo de medição.
- Uma medida consistente entre vários projetos e organizações.

Apresentando um pouco da história, segundo Vazquez, Simões e Albert (2005, p. 37), a técnica de análise de pontos por função surgiu na IBM, no início da década de 70. Allan Albrecht foi encarregado de medir a produtividade de vários projetos de software, desenvolvidos em diferentes linguagens, assim surgiu à necessidade de uma métrica que fosse independente da linguagem de programação utilizada. A técnica culminou em 1986, na fundação do International Function Point Users Group – IFPUG. Em 1990, o IFPUG publicou a versão 3.0 do Manual de Práticas de Contagem, que foi a primeira versão considerada madura. Embora posteriormente tenham surgido outras técnicas de medição funcional, a mais aceita é realizado pela análise dos pontos de função do IFPUG. Existem vários benefícios com a utilização de APF:

- Por ser independente de tecnologia, baseando-se nos requisitos de software para medição, torna-se uma ferramenta de normalização para comparação de software;

- Suporta a análise de qualidade e produtividade;
- Pode medir o tamanho em cada etapa do ciclo de desenvolvimento;
- Meio para estimar custo e recursos necessários para o desenvolvimento e manutenção do software.

APF é atualmente um instrumento utilizado por profissionais da área de sistemas e em empresas de todos os portes e segmentos da economia brasileira. Inicialmente o foco principal de sua aplicação era em estimativas, mas também tem sido aplicada, com êxito, como unidade de medição de contratos de desenvolvimento de software e como ferramenta na gerência de projetos. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 13)

É importante lembrar que APF é apenas uma técnica de medição, e com esta técnica é possível medir ou estimar um tamanho para um software a partir das funções requeridas ou incluídas no projeto. E após o tamanho ter sido estipulado ou calculado, se for de interesse, é possível orçar custo, tempo e/ou equipe de desenvolvimento.

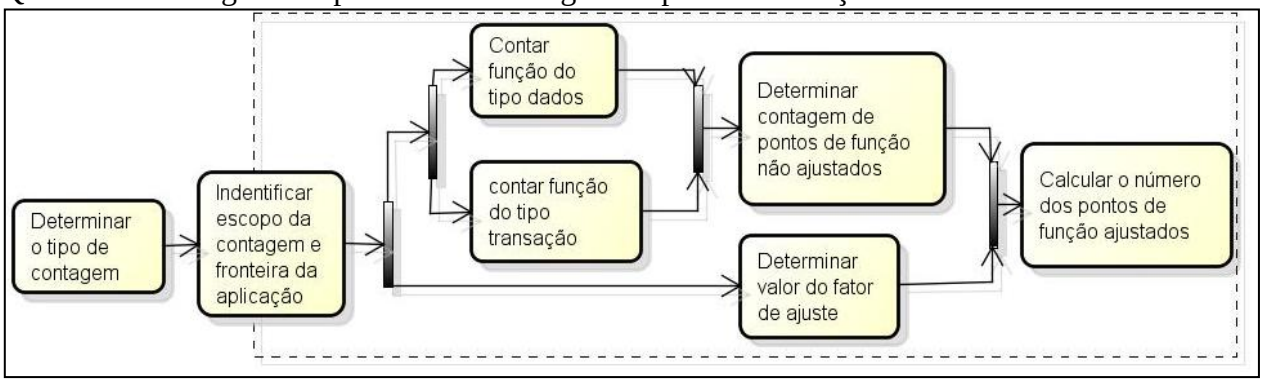
Os requisitos e a contagem de pontos de função

Segundo Vazquez, Simões e Albert (2005, p. 49), na análise de pontos por função, os requisitos funcionais são a base para o cálculo dos pontos de função não-ajustados e alguns requisitos não-funcionais são integrantes das Características Gerais de Sistema utilizadas na fase de determinação do fator de ajuste utilizado para o cálculo do número de pontos de função ajustados. Os requisitos são importantes, tanto para a determinação do tamanho do software, como também para os desenvolvedores, por isso eles devem identificar componentes técnicos úteis, porém devem ser descritos utilizando uma linguagem comum a todos os envolvidos no projeto.

Processo de contagem de pontos por função

Segundo Vazquez, Simões e Albert (2005, p. 44), para a determinação do tamanho funcional da aplicação ou projeto, a partir dos conceitos da técnica de contagem de pontos por função algumas abstrações se fazem necessárias. Estes conceitos representam os componentes do sistema que fornecem funcionalidades de armazenamento e processamento de dados aos usuários e por eles reconhecidas e solicitadas. A técnica também fornece definições de conceitos utilizados como referência na identificação desses componentes. O diagrama, do quadro 1, representa as etapas do processo de contagem, bem como suas relações de interdependência.

Quadro 1: Visão geral do processo de contagem de pontos de função.



Fonte: Vazquez, Simões e Albert (2005)

Antes de especificar o que é cada etapa do processo de contagem, faz-se necessário a definição de dois termos que serão usados posteriormente, usuário e aplicação.

Usuário: O conceito de usuário, para a análise de pontos por função, não se limita apenas a pessoas físicas que utilizam o software. Usuário pode ser qualquer coisa que interaja com o sistema ou especifique seus requisitos. Pois se a contagem de pontos por função fosse baseada apenas no conceito de usuário como sendo uma pessoa, não seria possível medir sistemas que não têm interface com o usuário final. Sendo que APF também pode ser aplicada nestas situações. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 52)

Aplicação: Para o IFPUG: "Uma aplicação é um conjunto coeso de dados e procedimentos automatizados que suportam um objetivo de negócio, podendo consistir de um ou mais componentes, módulos ou subsistemas". A aplicação deve ser definida de acordo com a

visão do usuário, sem considerar a arquitetura ou tecnologia utilizada no software. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 56)

Determinar o tipo de contagem

Para Vazquez, Simões e Albert (2005, p. 55), APF pode ser aplicada a projetos de desenvolvimento ou melhoria, e também em aplicações. Os três tipos de contagem podem ser vistas no quadro 2.

Quadro 2: Tipos de contagem em APF.

Projeto de desenvolvimento: o número de pontos de função de um projeto de desenvolvimento mede a funcionalidade fornecida aos usuários finais do software referentes à sua primeira instalação. A contagem também abrange as eventuais funções de conversão de dados necessárias à implantação do projeto.
Projeto de melhoria: o número de pontos de função de um projeto de melhoria mede as funções adicionadas, modificadas ou excluídas do sistema pelo projeto, e também as eventuais funções de conversão de dados.
Aplicação: o número de pontos de função de uma aplicação mede a funcionalidade fornecida aos usuários por uma aplicação instalada.

Fonte: adaptado de Vazquez, Simões e Albert (2005)

A Fronteira da Aplicação e o Escopo da Contagem

Para Vazquez, Simões e Albert (2005, p. 56), "A fronteira da aplicação é a interface conceitual que delimita o software que será medido e o mundo exterior (usuário)".

Determinar a fronteira da aplicação é determinar o que faz e o que não faz parte de cada aplicação em um software. Esta determinação tem grande importância para o processo de contagem, para delinear deve se levar em consideração a regra de negócio e não condições

técnicas como arquitetura e condições técnicas do software. As regras especificadas pelo IFPUG para determinar a fronteira de uma aplicação segundo Vazquez, Simões e Albert (2005, p. 56), são:

a) Sua determinação deve ser feita com base no Ponto de Vista do Usuário. O foco deve estar no que ele pode entender e descrever.

b) A fronteira entre aplicação deve ser baseada na separação das funções conforme estabelecido pelos processos de negócio, não em considerações tecnológicas.

c) Em projetos de melhoria, a fronteira estabelecida no início do projeto deve estar de acordo com a fronteira já estabelecida para a aplicação sendo modificada.

O escopo da contagem define as funções que realmente serão inclusas na contagem, se ela abrangerá um ou mais sistemas ou apenas parte de um sistema. O escopo pode abranger todas as funcionalidades disponíveis, funcionalidades efetivamente utilizadas pelo usuário, ou algumas funcionalidades específicas. O conceito de escopo de contagem é mais exercitado em projetos de melhoria, já que nem todas as funcionalidades serão alteradas. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 58)

Funções do tipo dados

De acordo com Vazquez, Simões e Albert (2005, p. 56), "As funções do tipo dados representam as funcionalidades fornecidas pelo sistema ao usuário para atender as suas necessidades de armazenamento de dados."

As funções do tipo dados são funcionalidades fornecidas ao usuário através da aplicação a fim de satisfazer as necessidades de armazenamento de dados tanto internos como externos. As funções do tipo dados podem ser divididas em dois grupos, Arquivos Lógicos Internos (ALI) e Arquivos de Interface Externa (AIE).

ALI: um grupo logicamente relacionado de dados ou informações de controle, identificável pelo usuário e mantido dentro da fronteira da aplicação sendo contada. Sua principal intenção é armazenar dados mantidos através de uma ou mais transações da aplicação sendo contada. Ex.: tabelas de DB atualizadas pela própria aplicação.
AIE: um grupo logicamente relacionado de dados ou informações de controle, identificável pelo usuário e mantidos fora da fronteira da aplicação sendo contada. Sua principal intenção é armazenar dados referenciados através de uma ou mais transações da aplicação sendo contada. Ex.: dados recuperados de um Web Server.

Quadro 3: Funções do tipo dados.

Fonte: adaptado de Vazquez, Simões e Albert (2005)

Cada função do tipo dado possui um nível de complexidade em PF, que é determinada por dois parâmetros, a quantidade de tipos de dados (campos) e a quantidade de tipos de registro (subgrupos de dados dentro do arquivo). (VAZQUEZ; SIMÕES; ALBERT, 2005)

A classificação com relação à complexidade é fornecida pela tabela 3, a seguir:

Tabela 3: Tabela de complexidade para funções do tipo dados.

	Tipos de dados		
Tipos de registros	**< 20**	**20 – 50**	**> 50**
1	**Baixa**	**Baixa**	**Média**
2 - 5	**Baixa**	**Média**	**Alta**
> 5	**Média**	**Alta**	**Alta**

Fonte: adaptado de Vazquez, Simões e Albert (2005)

Já a tabela 4, demonstra a contribuição dos pontos de função não ajustados das funções do tipo dado.

Tabela 4: Tabela de contribuição dos pontos de função não-ajustados das funções do tipo dado.

Tipo de Função	Baixa	Média	Alta
Arquivo Lógico Interno	7 PF	10 PF	15 PF
Arquivo de Interface Externa	5 PF	7 PF	10 PF

Fonte: adaptado de Vazquez, Simões e Albert (2005)

Funções do tipo transação

Para Vazquez, Simões e Albert (2005, p. 59), "As funções do tipo transação representam os requisitos de processamento fornecidos pelo sistema ao usuário". As funções do tipo transação podem ser vistas no quadro 4 abaixo.

Entrada Externa (EE): seu objetivo principal é atualizar os dados do sistema, ou também modificar o seu comportamento. Estes dados ou informações são estímulos recebidos externamente à fronteira da aplicação.
Saída Externa (SE): é uma transação que envia dados ou informações de controle para fora da fronteira da aplicação. Seu objetivo principal é apresentar informações que exigem lógica de processamento que não seja uma apenas uma simples recuperação de dados ou informações de controle. Seu processamento deve conter caçulo, ou criar dados derivados, ou manter um arquivo lógico interno, ou alterar o comportamento do sistema.
Consulta Externa (CE): seu objetivo é apenas recuperar e mostrar dados ao usuário, sem exigência de qualquer tipo de processamento. Uma CE recupera dados de um ALI ou AIE, e envia para fora da fronteira da aplicação.

Quadro 4: Funções do tipo Transação.

Fonte: adaptado de Vazquez, Simões e Albert (2005)

Assim como as funções do tipo dados, cada função do tipo transação também é classificada com relação a sua complexidade funcional (baixa, média, alta), determinada por dois parâmetros, a quantidade de tipos de dados e quantidade de arquivos referenciados. (VAZQUEZ; SIMÕES; ALBERT, 2005)

A classificação é demonstrada pelas seguintes tabelas 5 e 6, onde a tabela 5 demonstra a complexidade para Entradas Externas (EEs) e a tabela 6 a complexidade para Saídas Externas (SEs) e Consultas Externas (CEs).

Tabela 5: Tabela de complexidade para Entradas Externas (EEs).

	Tipos de Dados (TDs)		
Arquivos Referenciados (ARs)	**< 5**	**5 – 15**	**> 15**
< 2	**Baixa**	**Baixa**	**Média**
2	**Baixa**	**Média**	**Alta**
> 2	**Média**	**Alta**	**Alta**

Fonte: adaptado de Vazquez, Simões e Albert (2005)

Tabela 6: Tabela de complexidade para Saídas Externas (SEs) e Consultas Externas (CEs).

	Tipos de Dados (TDs)		
Arquivos Referenciados (ARs)	**< 6**	**6 – 19**	**> 19**
< 2	**Baixa**	**Baixa**	**Média**
2 – 3	**Baixa**	**Média**	**Alta**
> 3	**Média**	**Alta**	**Alta**

Fonte: adaptado de Vazquez, Simões e Albert (2005)

A tabela 7 demonstra as contribuições de pontos por funções não ajustados das funções do tipo transação.

Tabela 7: Tabela de contribuição dos pontos de função não-ajustados das funções do tipo transação.

Tipo de Funcao	Baixa	Média	Alta
Entrada Externa	3 PF	4 PF	6 PF
Saída Externa	4 PF	5 PF	7 PF
Consulta Externa	3 PF	4 PF	6 PF

Fonte: adaptado de Vazquez, Simões e Albert (2005)

Determinar contagem de pontos de função não ajustados

Segundo Vazquez, Simões e Albert (2005), os pontos de função não ajustados medem os requisitos específicos do usuário, e para obtermos, basta simplesmente depois de classificar cada função de acordo com seu nível de complexidade, somar o peso de cada uma das funções.

Determinar o valor do fator de ajuste

Para Vazquez, Simões e Albert (2005), o fator de ajuste tem o propósito de medir requisitos gerais da aplicação, e ajusta em ± 35% de acordo com as quatorze características gerais apresentadas pelo quadro 5 abaixo:

1. Comunicação de Dados	8. Atualização On-Line
2. Processamento Distribuído	9. Complexidade de Processamento
3. Performance	10. Reutilização
4. Configuração Altamente Utilizada	11. Facilidade de Instalação
5. Volume de Transações	12. Facilidade de Operação
6. Entradas de Dados On-Line	13. Múltiplos Locais
7. Eficiência do Usuário Final	14. Facilidade de Mudanças

Quadro 5: As 14 características para base de fator de ajuste.

Fonte: Vazquez, Simões e Albert (2005)

Cada uma das quatorze características possui um nível de influência, que pode variar em um intervalo de zero a cinco. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 118). Conforme o quadro 6 descreve.

0. Nenhuma influência
1. Influência Mínima
2. Influência Moderada
3. Influência Média
4. Influência Significativa
5. Grande Influência

Quadro 6: Níveis de influência das características gerais da aplicação.

Fonte: Vazquez, Simões e Albert (2005)

Depois de determinarmos os níveis de influências das características gerais, usamos a fórmula demonstrada pelo quadro 7 para calcular o fator de ajuste. Onde TDI é o somatório dos níveis de influência. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 118)

$$VAF = (TDI \times 0{,}01) + 0{,}65$$

Quadro 7: Fórmula para calcular o fator de ajuste.
Fonte: Vazquez, Simões, Albert, 2005, p. 118)

Devemos levar em consideração que quando a técnica de pontos de função foi apresentada por Allan Albercht, em 1979, não haviam as quatorze Características Gerais do Sistema (CGSs), sendo que o fator de ajuste poderia ser calculado de forma totalmente subjetiva. Somente mais tarde em 1984, foram introduzidas as atuais quatorze CGSs. No final do ano de 2002, o uso do fator de ajuste tornou-se opcional. (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 118)

Calcular o número dos pontos de função ajustados

Este é o último passo da contagem de pontos por função, a fórmula para calcular o número de os pontos de função ajustados vai variar para os três tipos de contagem: projeto de desenvolvimento, projeto de melhoria e aplicação. A fórmula para calcular os pontos por função ajustados de um projeto de desenvolvimento, pode ser vista no quadro 8.

Fórmula do Projeto de Desenvolvimento
DFP = (UFP + CFP) x VAF
DFP: Número de pontos de função do projeto de desenvolvimento. UFP: Número de pontos de função não ajustados das funções disponíveis após a instalação. CFP: Número de pontos de função não ajustados das funções de conversão. VAF: Valor do fator de ajuste.

Quadro 8: Fórmula para calcular os pontos de função ajustados para projeto de desenvolvimento.
Fonte: adaptado de Vazquez, Simões e Albert (2005)

A fórmula para calcular os pontos de função ajustados em um projeto de melhoria pode ser vista no quadro 9.

Fórmula do Projeto de Melhoria
EFP = [(ADD + CHGA + CFP) x VAFA] + (DEL X VAFB)
EFP: Número de pontos de função de projeto de melhoria. ADD: Número de pontos de função não ajustados das funções incluídas pelo projeto de melhoria. CHGA: Número de pontos de função não ajustados das funções modificadas. Reflete as funções depois das modificações. CFP: Número de pontos de função não ajustados adicionados pela conversão. VAFA: Valor do fator de ajuste da aplicação depois do projeto de melhoria. DEL: Número de pontos de função não ajustados das funções excluídas pelo projeto de melhoria. VAFB: Valor do fator de ajuste da aplicação antes do projeto de melhoria.

Quadro 9: Fórmula para calcular os pontos de função ajustados para projeto de melhoria.
Fonte: adaptado de Vazquez, Simões e Albert (2005)

Segundo Vazquez, Simões e Albert (2005, p. 137), em APF existem duas fórmulas para calcular o número de pontos de função da aplicação. Uma para a primeira contagem dos pontos de função da aplicação e outra para recalcular seu tamanho após um projeto de melhoria ter alterado sua funcionalidade. A fórmula da contagem inicial da aplicação pode ser visualizada no quadro 10.

Fórmula da Contagem Inicial da Aplicação
AFP = ADD + VAF
AFP: Número de pontos de função ajustados da aplicação. ADD: Pontos de função não ajustados das funções instaladas. VAF: Valor do fator de ajuste da aplicação.

Quadro 10: Fórmula para calcular os pontos de função ajustados para contagem inicial da aplicação.
Fonte: adaptado de Vazquez, Simões e Albert (2005)

Já a fórmula para calcular seu tamanho após um projeto de melhoria ter alterado sua funcionalidade, é exibida pelo quadro 11.

Fórmula da Aplicação após Projeto de Melhoria
AFP = [(UFPB + ADD + CHGA) – (CHGB + DEL)] x VAFA
AFP: Número de pontos de função ajustados da aplicação. UFPB: Pontos de função não ajustados da aplicação antes do projeto de melhoria. ADD: Pontos de função não ajustados das funções incluídas pelo projeto de melhoria. CHGA: Pontos de função não ajustados das funções alteradas pelo projeto de melhoria depois de seu término. CHGB: Pontos de função não ajustados das funções alteradas pelo projeto de melhoria antes de seu término. DEL: Pontos de função não ajustados das funções excluídas pelo projeto de melhoria. VAFA: Valor do fator de ajuste depois do projeto de melhoria.

Quadro 11: Fórmula para calcular os pontos de função ajustados para aplicação após projeto de melhoria.
Fonte: adaptado de Vazquez, Simões e Albert (2005)

3 ORIENTAÇÃO A OBJETOS

A palavra objeto tem um sentido muito amplo, que nos permite abstrair todas as coisas tanto físico como abstratas em objetos. Se pararmos e analisarmos em nossa volta, tudo pode ser traduzido em objetos, cadeiras, mesas, animais, pessoas, sentimentos, contas bancárias, e todas as outras coisas que conhecemos ou imaginamos. A Orientação a Objetos é a extração desses objetos para um software, preservando seu comportamento do mundo real. Sendo que após este objeto ter sido criado no mundo computacional, com todas as suas características e habilidades, basta usá-lo, sem se preocupar como ele realiza suas tarefas, apenas sabendo que ele as possui é o suficiente.

Sabemos que no mundo real os objetos interagem entre si, assim também deve ser nos Sistemas Computacionais. De acordo com Correia e Tafner (2001, p. 2), “Orientação a Objetos consiste em considerar os Sistemas Computacionais como uma coleção de objetos que interagem de maneira organizada.”

Objetos Computacionais são simplificações dos Objetos Reais. Os Objetos Computacionais devem interagir dentro de um sistema de computador da mesma forma como os

Objetos Reais interagem no mundo real, para isto é importante que estes objetos saibam como reagir perante uma ação. Não é o sistema que deve determinar o comportamento de um Objeto Computacional, mas ele próprio deve possuir a informação necessária para saber como comportar-se diante uma situação. Sintetizando, Objetos Computacionais são estruturas de programação que contém as informações e os comportamentos que representam um objeto dentro de um sistema. (CORREIA; TAFNER, 2001)

Segundo David (2007), a programação orientada a objetos tenta aproximar mundo real do computacional, sendo que a idéia principal é simular o mundo real dentro do computador. O programador é responsável por moldar o mundo dos objetos, e explicar para estes objetos como eles devem interagir entre si. Os objetos se comunicam através de mensagens, o programador deve especificar as mensagens que cada objeto pode receber, e de que forma ele deve reagir a estas mensagens.

Um software desenvolvido usando programação orientada a objetos possui uma série de vantagens, é mais organizado, torna mais fácil a sua manutenção e reutilização de código, além de menor risco de bugs. Uma das principais vantagens é poder usar a mesma metodologia tanto na análise como na programação. Quando usada uma equipe grande de desenvolvimento, torna mais fácil gerenciar o desenvolvimento deste software, pois a divisão do código é mais lógica e melhor encapsulada, permitindo dividir o projeto em partes, como pacotes e classes, sendo possível dividir estas partes entre os membros.

3.1 ALGUNS PRINCÍPIOS DA ORIENTAÇÃO A OBJETOS

3.1.1 Abstração

Segundo Correia e Tafner (2001, p. 11), o termo abstração, aplicado ao desenvolvimento de sistemas, significa representar no sistema computacional, apenas o necessário, o que realmente será útil. Significa isolar os objetos que nos interessam do mundo complexo em que se situam, e neles representar somente as características que são relevantes para o problema em questão.

Um mesmo objeto do mundo real, por exemplo, o objeto pessoa, pode ser representado diferentemente, de acordo com o problema em questão. Em um sistema para área de odontologia, o objeto pessoa teria necessariamente atribuído a ele a característica “dentes”, porém, em um software para folha de pagamento, esta informação poderia ser irrelevante.

3.1.2 Encapsulamento

Na Orientação a Objetos os dados e os processos que tratam esses dados estão “encapsulados” numa única entidade. A única maneira de conhecer e alterar os atributos de um objeto é através de seus métodos. O encapsulamento disponibiliza o objeto com toda a sua funcionalidade, sem que você precise saber como ele funciona internamente, nem como armazene internamente os dados que você recupera. Outra vantagem é permitir que você faça modificações internas em um objeto, acrescentando métodos sem se preocupar com os outros componentes do sistema que utilizam este mesmo objeto. (CORREIA; TAFNER, 2001)

Também segundo Correia e Tafner (2001, p. 11), o que importa para poder haver interação entre dois objetos é que um conheça o conjunto de operações disponíveis do outro (interface) para que então envie e receba informação, ou mesmo ordene a realização de procedimentos.

O encapsulamento é uma das grandes vantagens da Programação Orientada a Objetos em relação à Estruturada, torna o desenvolvimento e principalmente a manutenção de software mais fácil, pois a codificação fica mais organizada e dividida, ficando também mais fácil de entender. Contando ainda com a principal vantagem de se poder alterar radicalmente parte de um software, sem que haja impacto no restante do programa.

3.1.3 Objetos

Quando falamos em objetos, pensamos em representações do mundo real, ou seja, do mundo físico tal qual conhecemos, segundo Correia e Tafner (2001, p. 13), esta é uma boa maneira de começar a pensar de forma Orientada a Objetos. Porém, é através de exemplos que se entende melhor o sentido de um objeto neste estilo de programação, sendo assim tomaremos como exemplo o objeto carro.

Começaremos definindo características próprias deste objeto, o carro que poderiam ter entre outras: um nome, um modelo, um fabricante, uma cor, um ano de fabricação. Como já descrito acima, as informações relevantes em relação a um objeto, podem variar dependendo do problema em questão, mas por motivo de melhor entendimento aqui tomamos apenas características mais simples. Estas características que definem um objeto são chamadas de "atributos". Segundo Correia e Tafner (2001, p. 15), atributos dos objetos são "variáveis" ou "campos" que armazenam os diferentes valores que as características dos objetos podem conter.

Além de atributos, um objeto pode possuir ações, um carro, possui por exemploo, a capacidade de acelerar e frear, para este conjunto de ações dá-se o nome de "métodos", que inclusive é a única forma de interagir com os objetos, seguindo o conceito de encapsulamento. Segundo Correia e Tafner (2001, p. 16), os métodos são as ações que o objeto pode realizar.

Contudo, um objeto é formado por atributos, que são suas características, e métodos, que são suas ações.

3.1.4 Classes

Segundo David (2007), uma classe é uma abstração que define um tipo de objeto e o que objetos deste determinado tipo têm dentro deles (seus atributos), e também define que tipo de ações esse tipo de objeto é capaz de realizar (métodos).

Para Correia e Tafner (2001, p. 17), “[...] a classe é um modelo, e todos os objetos daquela classe têm os mesmos atributos (embora esses atributos possam ter valores diferentes) e os mesmos métodos”.

Uma classe tem por finalidade criar objetos de um determinado tipo, por exemplo, um objeto carro, para criá-lo, precisamos de uma classe molde com atributos como modelo, marca, fabricante, placa, cor, proprietário e os métodos como acelerar e frear. A partir deste molde, seria possível criar um objeto carro.

3.1.5 Hierarquia de Classes

Um conceito importante da Orientação a Objetos é hierarquia de classes. Para entendermos vamos observar a figura 1, abaixo.

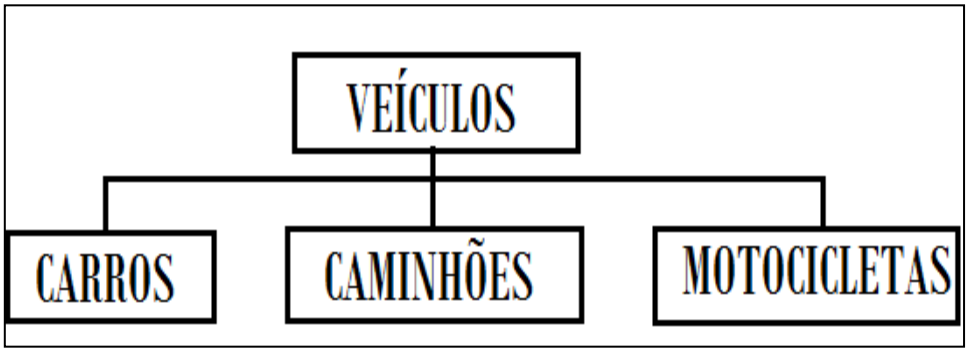


Figura 1: Hierarquia de Classes

Para criar um objeto carro, caminhão, ou motocicleta, necessitamos de uma classe respectivamente para cada um desses objetos, porém estes três elementos possuem características em comum, pois são membros de uma categoria maior, os veículos. Sendo assim, se criarmos uma classe pai veículos, podemos omitir nas classes filhos carros, caminhões, motocicletas, tudo que já foi atribuído a classe pai veículos.

Segundo Correia e Tafner (2001, p. 18), chamamos de “ancestrais” as classes das quais as outras dependem, e de “descendentes” as classes originadas a partir de outras.

3.1.6 Instanciação

Para definir instanciação recorremos a Correia e Tafner (2001, p. 18), “A instanciação acontece quando a classe produz um objeto, como se fosse uma espécie de modelo ou gabarito para a criação de objetos. Conforme a teoria da Orientação a Objetos, dizemos que um objeto é nada mais nada a menos, que a instância de uma classe.”

Uma classe serve somente de molde para criar objetos, somente os objetos têm existência. Quando instanciamos criamos um novo objeto.

3.1.7 Herança

De acordo com Correia e Tafner (2001, p. 35), “Herança significa que todos os atributos e métodos programados no ancestral já estarão automaticamente presentes em seus descendentes sem necessidade de re-escrevê-los.”

A herança permite que uma classe herde características, métodos e atributos de outra classe. Assim, além da herança ser um mecanismo inteligente de economia de código também significa maior integridade, pois quando se altera um comportamento em uma classe, todas as classes descendentes desta também estarão utilizando o método atualizado sem necessidade de reprogramação. (CORREIA; TAFNER, 2001)

Herança Simples

A herança é denominada simples quando uma classe herda apenas características de apenas uma superclasse. Entretanto, uma mesma superclasse ou classe ancestral, pode gerar mais de uma subclasse, ou classe filha. (CORREIA; TAFNER, 2001)

Herança Múltipla

A herança é denominada múltipla quando uma classe herda características de duas ou mais superclasses. Na herança múltipla assim como na herança simples, uma mesma superclasse, pode gerar mais de uma subclasse. (CORREIA; TAFNER, 2001)

Classes abstratas

Segundo a definição de David (2007):

> [...] uma classe abstrata é um conjunto de informações a respeito de uma coleção de outras classes. Uma classe abstrata sozinha é completamente inútil, já que não podemos instanciar um objeto desta classe, podemos apenas instanciar objetos de classes que estendem a classe abstrata inicial. Ela serve apenas para simplificar o sistema, juntando em um único lugar diversas características que são comuns a um grupo de classes.

Isto quer dizer que jamais podemos criar um objeto do tipo animal, uma classe animal serviria apenas para agrupar informações de suas classes descendentes, gatos, cães, cavalos.

3.1.8 Polimorfismo

Para Correia e Tafner (2001, p. 35), "O polimorfismo ocorre quando um método que já foi definido no ancestral é redefinido no descendente com um comportamento diferente."

Para David (2007), polimorfismo na Orientação a Objetos, significa que um mesmo tipo de objeto, sob certas condições, pode realizar ações diferentes ao receber uma mesma mensagem. Ou seja, apenas olhando o código fonte não sabemos exatamente qual será a ação tomada pelo sistema, sendo que o próprio sistema é quem decide qual método será executado, dependendo do contexto durante a execução do programa.

Sempre que um método for reescrito na classe descendente, será executado o código da classe descendente, e quando ele não for reescrito, ele será executado a partir da classe pai. Desta forma, uma mesma mensagem, por exemplo, “fale" enviada a um objeto da classe Animal (descendente) pode ser interpretada de formas diferentes, dependendo do objeto em questão. Se o objeto em questão for um cachorro, a resposta deste objeto poderia ser um latido, se fosse um gato poderia ser um miado.

4 UML

A Unified Modeling Language (UML), segundo Penter (2004, p.3), é uma destilação de três notações principais e uma série de técnicas de modelagem retiradas de metodologias bastante diversificadas, que já existem na prática desde as duas últimas décadas. E desde o início de seu desenvolvimento formal, em 1994, teve um impacto inegável na forma como vemos o desenvolvimento de sistemas, tornado-se padrão para a modelagem de softwares em quase 70% dos centros de TI. A UML é formada por mais de 50 ferramentas de modelagem comercias e acadêmicas, que oferecem suporte à modelagem de software e à modelagem de negócio.

Uma definição bem resumida de UML, descrita por Martin Fowler (2005): “[...] é uma família de notações gráficas, apoiada por um meta-modelo único, que ajuda na descrição e no projeto de sistemas de software, particularmente daqueles construídos utilizando o estilo orientado a objetos (OO)”.

4.1 DIAGRAMA DE CLASSE

> Um diagrama de classes descreve os tipos de objetos presentes no sistema e os vários tipos de relacionamentos estáticos existentes entre eles. Os diagramas de classes também mostram as propriedades e as operações de uma classe e as restrições que se aplicam à maneira como os objetos estão conectados. (FOWLER, 2005)

Os diagramas de classes são os diagramas principais e provavelmente mais utilizados da UML. Também é o diagrama principal para gerar código, além de muitas vezes ser a base para a construção de outros diagramas.

Segundo Pender (2004), o diagrama de classes está no núcleo do processo de modelagem de objetos. Ele modela as definições de recursos essências à operação correta do sistema. Todos os outros diagramas de modelagem descobrem informações sobre estes recursos que por fim precisam ser encaminhados ao diagrama de classes. Este diagrama modela os recursos usados para montar e operar o sistema, estes recursos representam pessoas, materiais, informações e comportamentos. O diagrama de classes cada recurso em termos de sua estrutura, relacionamentos e comportamentos. Este diagrama é também a origem para a geração de código.

4.1.1 Propriedades

Segundo Fowler (2005), as propriedades representam as características estruturais de uma classe. Superficialmente elas podem ser consideradas como campos de uma classe. As propriedades aparecem em duas notações bastante distintas: atributos e associações. Estas duas notações, embora bastante distintas em um diagrama, na realidade elas são a mesma coisa.

Atributos

A notação atributo é bastante simples, cada propriedade é descrita por uma linha dentro da caixa de classe em si. Para poder representar a visibilidade dos atributos e operações em uma classe utiliza-se os seguintes marcas e significados:

+ público: visível em qualquer classe.

protegido: visível por qualquer descendente.

- privado: visível somente dentro da classe.

É necessário também dar um nome ao atributo. A denominação deve representar como a classe se refere a este atributo.

Também é necessário estabelecer um tipo para este atributo. O tipo do atributo é uma restrição de objeto que pode ser colocado no atributo.

Um exemplo de representação de atributo pode ser: “- nome: String”.

Associação

Os objetos se relacionam uns com os outros, e estas relações devem ser representadas no diagrama de classes. A associação é uma destas representações.

Segundo Fowler (2005), uma associação é representada por uma linha cheia entre duas classes, direcionada da classe de origem para a classe de destino. O nome da propriedade fica no destino final da associação, junto com sua multiplicidade. O destino final da associação vincula a classe à classe que é o tipo da propriedade.

Segundo Pender (2004) o nome de uma associação é muito importante, e deve expressar ao leitor a intenção do relacionamento. Se o nome for vago, ele introduz confusão e debate, aumentando o custo e reduzindo a eficiência no processo de modelagem.

Figura 2: Notação de associação

Agregação

Segundo Penter (2004), a agregação descreve um tipo especial de associação, criado para nos ajudar a lidar com a complexidade. Em uma associação simples, nenhuma classe é superior a

outra, cada classe permanece independente da outra, elas simplesmente se comunicam. Já na agregação é definido um tratamento hierárquico definitivo.

Figura 3: Notação de agregação.

Composição

Segundo Penter (2004), a composição é usada para agregações onde o tempo de vida do objeto membro depende do tempo de vida do agregado. O agregado não apenas tem controle sobre o comportamento do membro, mas também tem controle sobre a criação e a destruição do membro. O objeto membro não pode existir em separado do agregado. Verifica-se na figura 4, notação de composição.

Figura 4: Notação de composição

Multiplicidade

Segundo Fowler (2005), multiplicidade de uma propriedade é uma indicação de quantos objetos podem preencher a propriedade. As multiplicidades mais usadas são mostradas pela tabela 8, a seguir.

Tabela 8: Representação de multiplicidades em UML.

1	exatamente 1 (um pedido dever ter exatamente um cliente)
0..1	pode ser 0 ou 1 (Um cliente corporativo pode ter ou não um representante de vendas)
*	(Um cliente não precisa fazer um pedido, não existe nenhum limite de pedidos para este cliente).

Fonte: adaptado de Fowler (2005).

5 RUBY

Ruby é uma linguagem de programação criada por Yukihiro “Matz” Matsumoto. Esta linguagem possui uma ampla quantidade de material tanto na Internet quanto em livros, o objetivo desta seção é expor resumidamente sua filosofia e características.

5.1 FILOSOFIA DO RUBY

Segundo Flanagan e Matsumoto (2008, p. 2), a filosofia do guia de Yukihiro Matsumoto conhecido também como Matz, para o design do Ruby é sumarizada em uma de suas citações: “O Ruby foi feito para a felicidade dos programadores”. Matz conhecia muitas linguagens, mas não estava satisfeito com nenhuma delas, então decidiu criar a sua própria linguagem que o satisfizesse como programador, e afirma, que para sua surpresa, muitos programadores sentiam o mesmo que ele, ficaram felizes quando descobriram e programaram no Ruby. Matz queria criar uma linguagem mais rápida e fácil, por isso todas as suas características são designadas a funcionarem com programadores comuns.

Ruby é uma linguagem elegante, fácil de aprender e usar, proporciona agilidade no processo de desenvolvimento, pois sua codificação é agradável e intuitiva.

5.2 CARACTERÍSTICAS DO RUBY

Ruby é uma linguagem de script, totalmente orientada a objetos, onde qualquer valor é um objeto, até mesmo classes e tipos primitivos, e onde toda função é um método. Seguindo o conceito de herança da orientação a objetos, algumas linguagens permitem heranças múltiplas, porém, Ruby propositalmente não aceita.

No Ruby, sempre que necessitamos de um conjunto de métodos, constantes ou variáveis de classe, sendo que a programação objeto-orientada não for necessária, podemos desfrutar de

módulos, que diferente de uma classe, não pode ser instanciado e nem subclassificado. A criação de um módulo é semelhante à de uma classe, apenas ao invés de usarmos a palavra class, usamos a palavra module. Os módulos podem ser usados como namespaces, e como misturadores. Como namespaces utilizamos diretamente seus métodos, que já foram programados para funções específicas. Como misturadores, chamados de módulos mixins, utilizando a palavra-chave include, vários módulos podem ser misturados numa mesma classe.

Ruby é uma linguagem de tipagem dinâmica e forte. Não é necessário declarar tipo para as variáveis, elas assumem um tipo automaticamente de acordo com o valor que lhe for dado, porém, não é possível somar uma cadeia de caracteres (String) com um valor inteiro (Fixnum), por exemplo.

Segundo Flanagan e Matsumoto (2008, p. 2), o Ruby tem uma complexa, mas expressiva gramática e uma biblioteca de classe integrada com um rico e poderoso API. O Ruby se inspira no Lisp, Samalltalk e Perl, mas usa uma gramática que é fácil para os programadores em C e JavaTM aprenderem. O Ruby é uma linguagem também apropriada para procedimentos e formatações de programação funcional. Isto inclui poderosas capacidades de metaprogramação e pode ser usada para criar linguagem de domínio específico ou DSLs.

6 RUBY ON RAILS

Ruby on Rails é um framework para o desenvolvimento de aplicativos Web, criado por David Heinemeier Hansson. Ruby on Rails ou simplesmente Rails é gratuito, de código aberto e desenvolvido na linguagem de programação Ruby.

Segundo Tate e Hibbs (2006), o desenvolvimento com Rails é muito mais simples, com menor quantidade de linhas de código, resultando em menos manutenção e tempo de desenvolvimento menor. Ruby on Rails, já revolucionou o desenvolvimento de aplicações web e facilitou a vida de centenas de milhares de desenvolvedores.

Para conhecermos mais sobre o Rails, recorremos às palavras do próprio criador David Heinemeier Hansson:

> O Rails é mais que uma estrutura de programação para criar aplicativos Web; é, também, uma estrutura para pensar sobre os aplicativos Web. Ele não funciona como uma lousa em branco tolerante a todo tipo de expressão, ao contrário, ele troca esta flexibilidade pela conveniência "do que a maioria das pessoas precisa na maior parte do tempo para fazer grande parte das coisas". É a camisa de força do designer evitando que ele se concentre nas coisas que simplesmente não importam e volte sua atenção para as que importam. (FERNANDEZ, 2008).

Através da declaração de David entende-se por que os projetos desenvolvidos com Rails são robustos e de codificação elegante. Ele traz uma estrutura agradável e elegante, que primeiramente precisamos entender, para depois programar com consistência, percebendo seu funcionamento, e aprendendo a desfrutar de suas convenções, utilizando-as ao nosso favor.

Na verdade o Rails junta às funcionalidades de outros cinco frameworks: Active Record, Action Pack, Action Mailer, Active Support e Active WebServices. Mesmo que cada framework tenha sua função independente, esta união é transparente para o desenvolvedor, pois o Rails os organiza e faz com que todos trabalhem de forma conjunta. Os dois frameworks principais são o Active Record e o Action Pack, pois são a sustentação do Desing Pattern de MVC (Model-View-Controller).

6.1 ACTIVE RECORD

Active Record é um dos frameworks que compõem o Rails, este pacote implementa parte do Desing Pattern de ActiveRecord, ele faz o mapeamento ORM (Operational Relational Mapping) de forma simples e transparente ao desenvolvedor. Este framework tem o trabalho de mapear tabelas para classes, e as colunas para atributos desta classe. O que significa que cada linha de uma determinada tabela, representa um objeto. A figura 5 representa um mapeamento da tabela produto para classe Produto.

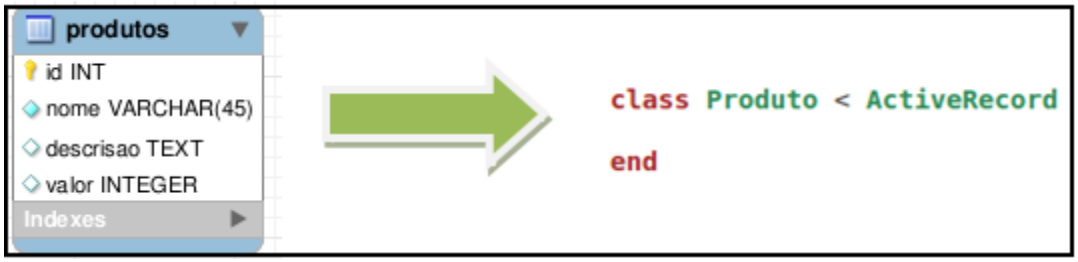


Figura 5: Mapeamento com Active Record

A classe Produto herda da classe ActiveRecord, assim todas as funcionalidades do Active Record serão herdas. Fazendo-se desnecessária a declaração de atributos para a classe Produto, nem mesmo os métodos de acesso (gets e sets), pois eles são criados automaticamente via meta-programação pelo Active Record. Além dos métodos de acesso outros métodos também são criados, como exemplo métodos para persistir (produto.save) e recuperar (produto.find(:id)).

Mas o framework é muito mais do que isto, existem uma série de funcionalidades que abrangem validações e relacionamentos entre classes. Contando com a convenção poucas configurações são necessária para o framework entrar em funcionamento, com a configuração de apenas um arquivo “database.yml”, toda aplicação estará conectada ao Banco de Dados (DB).

6.2 ACTION PACK

Pela camada controladora e a camada de visão estarem intimamente ligadas no Rails, elas são empacotadas em um único pacote, o Action Pack. Ele é responsável por fornecer dados aos usuários, e receber e encaminhar as requisições vindas dos usuários.

Segundo Akita (2006), pode-se dizer que o Action Pack é a espinha dorsal do mecanismo MVC (Model-View-Controller). Enquanto o Model é controlado pelo pacote ActiveRecord, as partes View e Controller são de responsabilidade do ActionPack. Ele é o maestro no despacho adequado de chamadas a métodos, redirecionando páginas e controlando o fluxo de ações pelo aplicativo.

Para melhor entender todo este mecanismo, deve-se observar a figura 4, presente na próxima página.

6.3 MVC

Segundo Helman (2007 apud LISBOA, 2008), MVC é a sigla de Model-View-Controller (Modelo-Visão-Controlador), e é um modo de quebrar uma aplicação em três partes, correspondentes a letra que formam a sigla. Originalmente, o MVC foi desenvolvido para mapear os tradicionais papéis entrada, processamento e saída para o reino GUI (Grafical User Interface). Com MVC a entrada do usuário, a modelagem do mundo externo e a resposta visual ao usuário são separadas e mantidas por objetos modelo, visualizador e controlador.

MVC é um desing pattern muito clássico, referente à arquitetura do software. A codificação de um software desenvolvido usando este padrão de desenvolvimento é organizada, legível, elegante e menos acoplada, ficando mais fácil de manter este software.

Segundo Helman (2007 apud LISBOA, 2008), o controlador, interpreta as entradas externas, transformando-as em comando que são enviados para o visualizador e/ ou modelo, para efetuar as mudanças apropriadas. O modelo gerencia um ou mais elementos de dados, responde a consultas sobre seu estado e a instruções de mudança de estado. O visualizador ou visão é responsável pelo mapeamento gráfico para um dispositivo, isto quer dizer que ele controla logicamente a apresentação ao usuário.

O Ruby on Rails é em um framework MVC, assim todo projeto que fizer seu uso seguirá a arquitetura divida em modelos, visões e controladores. Significando que a maior parte do código e do esforço do projeto estará nesta estrutura.

Como o Ruby on Rails é a junção de vários frameworks, esta parceria de frameworks tornou possível o funcionamento da arquitetura MVC, o Active Record, responsável pela comunicação com o DB, e o Action Pack que compreende a camada controladora e a camada de visualização.

O modelo MVC do Ruby on Rails é organizado, e as três camadas funcionam separadamente, sendo que somente entram em ação quando receberem uma requisição de outra camada, como mostrado na figura 6.

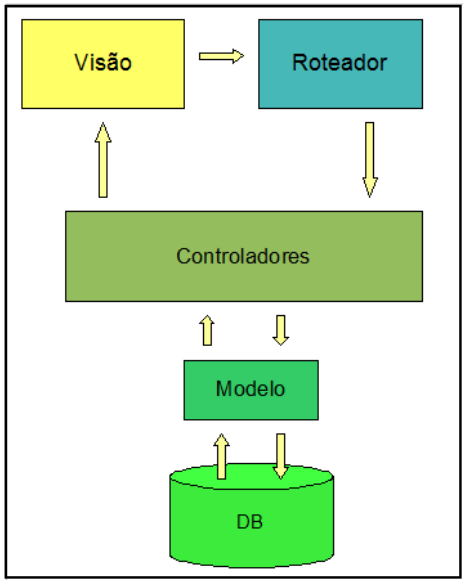


Figura 6: Fluxo de requisições MVC do Rails

6.3.1 Modelos

No *model* ou no português modelo, aplica-se toda regra de negócio de uma aplicação, ela é programada através de relacionamentos entre classes, validações e métodos. Esta camada também faz uma interface com o DB, qualquer interação que acontece com o DB ocorre através desta camada. Algumas vezes uma classe é apenas o mapeamento de uma tabela do DB, onde seus atributos representam as colunas desta tabela, neste caso esta classe herda de ActiveRecord, porém, nem todas as classes tem esta característica. O quadro 12 exibe um exemplo de modelo Ruby on Rails.

```
1  class FunctionalRequirement < ActiveRecord::Base
2
3
4  #Relacionamentos
5
6  belongs_to :project
7
8  has_many :non_functional_requirements
9
10
11
12 #Validações
13
14 validates_length_of  :name, :maximum=>100, :message=>" não pode exceder 100 caracteres!"
15
16 validates_length_of  :description, :maximum=>1000, :message=>" não pode exceder 1000 caracteres!"
17
18 validates_presence_of :name, :message=>" não pode ser vazio!"
19
20 validates_presence_of :project_id
21
22 #metodos
23 def self.human_class_name
24   "FunctionalRequirement"
25 end
26
27 end
```

Quadro 12: Exemplo de modelo Ruby on Rails

A classe representada pelo quadro 12, tem nome FunctionalRequirement, e estende de ActiveRecord. Na linha 6 acontece um relacionamento com a classe Project, “belongs_to: project” relacionamento muitos para um, ou seja, um objeto do tipo Project, poderá ter um lista de objetos do tipo FunctionalRequirement, por outro lado um objeto do tipo FunctionalRequirement pertence a um objeto do tipo Project. Na linha 8 acontece um relacionamento com a classe NonFunctionalRequirement, “has_many: non_functional_requirement” relacionamento um para muitos, ou seja, um objeto do tipo FunctionalRequirement poderá ter um lista de objetos do tipo NonFunctionalRequirement, por outro lado um objeto do tipo NonFunctionalRequirement pertence a um objeto do tipo FunctionalRequirement.

Continuando, nas linhas 14 e 16, são validações para tamanho, portanto validando o tamanho máximo para os campos name e description. Ao tentar persistir um objeto do tipo FunctionalRequirement que tiver os atributos name e description com um tamanho que ultrapasse 100 e 100 caracteres respectivamente, o objeto não será persistido, e as mensagens “não pode exceder 100 caracteres!” e “não pode exceder 1000 caracteres!” serão retornadas. Nas linhas 18 e

20, são validações de presença, os campos name e projeto devem ser presentes. Da linha 23 a 25 é um exemplo de método, que retorna o nome da classe.

6.3.2 Controladores

Segundo Thomas e Hansson (2008), o controlador Rails é o centro lógico do aplicativo, ele coordena a interação com o usuário, as visões e o modelo. O controlador também de vários serviços importantes:

- Ele é o roteador das requisições externas para ações internas. Ele transforma URLs amigáveis extremamente bem.
- Ele gerencia o cache, o que fornece aos aplicativos um aumento de desempenho formidável.
- Ele gerencia módulos auxiliares, o que estende a capacidade dos templates de visão sem aumentar o volume de código.
- Ele gerencia sessões, dando aos usuários a impressão de uma interação contínua com o aplicativo.

6.3.3 Visão

Segundo Thomas e Hansson (2008), a visão no Rails, é responsável por criar toda ou parte de uma página a ser exibida em um navegador. Em geral a página é criada por duas partes, a parte fixa, criada com fragmentos de código HTML, e a parte dinâmica, criado pelo método de ação no controlador.

6.4 ESTRUTURA DE DIRETÓRIOS

Segundo Thomas e Hansson(2008), o Rails supõe certo layout de diretórios em tempo de execução.

A figura 7 mostra a estrutura de diretório de primeiro nível, gerada pelo Rails 2.3.11. “rails my_app”.

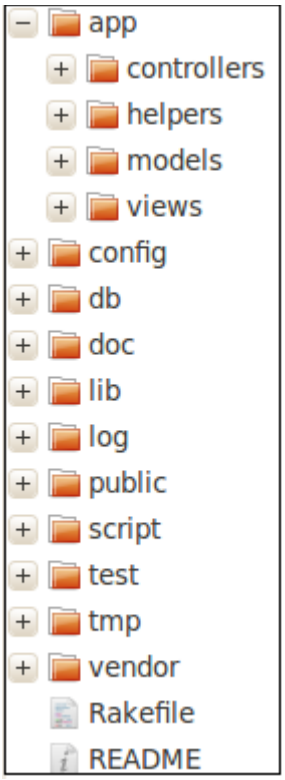


Figura 7: Estrutura de diretórios gerada pelo Rails 2.3.11.

A tabela 9 mostra cada diretório que é gerado pelo Rails 2.3.11, alinhado juntamente a sua respectiva finalidade.

Tabela 9: Especificando os diretórios e arquivos gerados pelo Rails.

Diretório/Arquivo	Descrição
app/controllers	Detém os controladores, eles devem ser nomeados com sufixo _controller, e extensão .rb (FunctionalRequiremente_controller.rb), para mapeamento de URL automatizada. Todos os crontroladores devem descender de ApplicationController, que por sua vez descende de ActionController::Base.
app/models	Possui modelos que devem ser nomeados com extensão .rb. A maioria dos modelos descenderá de ActiveRecord::Base.
app/view	Contém os arquivos para visão, e devem ser nomeados como functional_requirements/índex.html.erb, para a ação índex do controlador FunctionalRequiremet_controller.rb.
app/view/layouts	Contém os arquivos de modelo para layout, que são usados para visão. Modelos comumente definidos para cabeçalho/rodapé, e corpo da página.
app/helpers	Contém arquivos com métodos ajudantes. Eles podem ser usados para extrair métodos das visões.
config/	Arquivos de configuração para o ambiente Rails, o mapa do roteador, o banco de dados e outras dependências.
db/	Contém o esquema de banco de dados em schema.rb. Arquivos de migrações dentro de db/migrate.
lib/	Bibliotecas específicas da aplicação. Basicamente qualquer tipo de código que não se enquadra em controllers, models, ou helpers.
public/	O diretório diponível para o servidor Web, contem subdiretórios para imagens, folhas de estilo e javascripts. Contém também os arquivos HTML padrão.
script/	Scripts ajudantes para automação e geração.
test/	Unidades de testes funcionais.
vendor/	Bibliotecas externas que a aplicação depende. Plugins e gems.
README	É usado para descrição da aplicação.
Rakefile	Pode ser usado para executar teste, criar documentação, extrair a estrutura atual do esquema da aplicação, etc.

7 REDMINE

Redmine é uma aplicação web para gerenciamento de projetos de software, desenvolvido na linguagem de programação Ruby, juntamente com o framework Rails. É uma ferramenta que vem ganhando grande espaço, pois além de ser simples e interativo possui Licença Pública Geral (GPL 2.0).

O projeto Redmine é gerenciado por Jean Philippe Lang, a página oficial do projeto pode ser encontrada no endereço eletrônico http://www.redmine.org. Uma demonstração on-line é encontrada no endereço: http://demo.redmine.org/.

Alguns dos recursos oferecidos pelo projeto Redmine são:

- Suporte a diversos Bancos de Dados (MySQL, PostgreSQL, SQLite);
- Integração com repositórios (SVN, CVS, Git, Mercurial, Bazaar e Darcs);
- Suporte a múltiplas linguagens;
- Suporte a múltiplos projetos;
- Controle de tarefas;
- Calendários e gráficos;
- Notícias, documentos e arquivos (gerenciamento);
- Wiki e fórum;
- Suporte a feeds e documentação por e-mail;
- Uso de plugins.

A figura 8 a seguir mostra todas as opções da barra de menus, depois que um projeto é gerado.

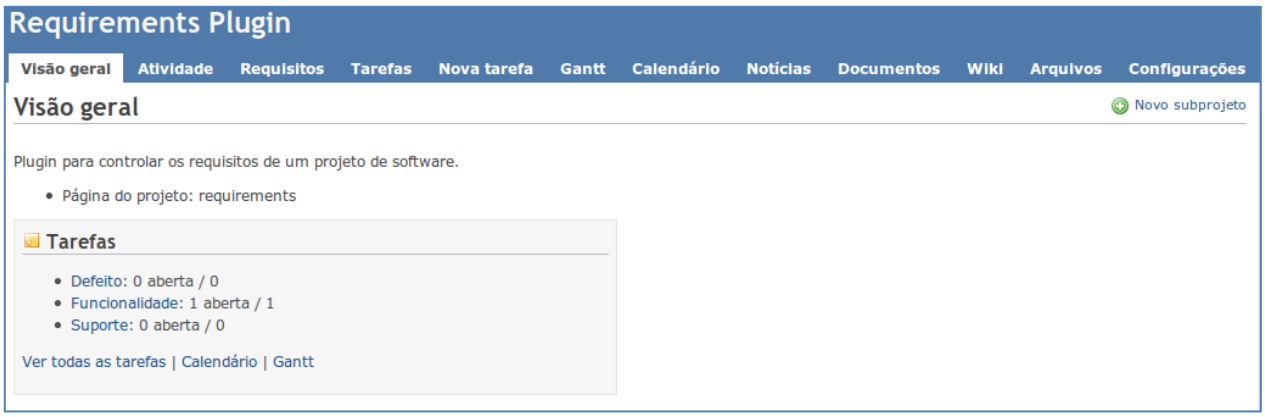


Figura 8: Página inicial de um projeto no Redmine

7.1 INSTALAÇÃO

Além de todas as vantagens do gerenciador de projetos Redmine, ele é fácil de instalar e configurar. A instalação pode ser feita tanto em um servidor na Internet, como também em um servidor de rede local, ou em um micro pessoal.

Demonstra-se aqui a instalação do Redmine versão 1.2, no sistema operacional Ubuntu 10.4, instalado em um computador pessoal. A documentação oficial da instalação é encontrada em: http://www.redmine.org/projects/redmine/wiki/RedmineInstall.

A linha de comando demonstrada no quadro 13, abaixo, instala todas as dependências necessárias, caso algumas delas já estejam instadas, apenas notificações serão geradas.

```
sudo aptitude install build-essential sudo apt-get install ssh openssh-server
mysql-server phpmyadmin rails rubygems mongrel ruby1.8-dev libmysql-ruby
subversion apache2 ruby irb rdoc rake libapache2-mod-fastcgi
```

Quadro 13: Linha de comando para instalar as dependências para o Redmine.

Nesta instalação utiliza-se o DB MySQL. Durante a instalação da dependência mysql-server, ou então Servidor MySQL, será requisitado um usuário e uma senha, basta completar cada campo, e posteriormente selecionar a opção <ok>.

Com as dependências instaladas, devem ser verificadas as versões do framework Ruby on Rails e do framework rake. Caso as versão não sejam a 2.3.11 e 0.8.7 respectivamente, devem ser executados os comandos: “gem install rails –v=2.3.11”, “gem unistall rake” e “gem install –v=0.8.7 rake”. Outra gem que é requerida é Rack “gem install rack –v=1.1.1”.

Para deixar o DB pronto para o Redmine, como usuário administrador do MySQL versão 5.1.41, insere-se as linhas do quadro abaixo na linha de comando do MySQL.

```
create database redmine character set utf8;
create user 'alexandro'@'localhost' identified by 'root';
grant all privileges on redmine.* to 'alexandro'@'localhost' identified by 'root';
```

Quadro 14: Criando e configurando base de dados e usuário MySQL, para o Redmine.

Nas linhas do quadro 14, acima, primeiramente cria-se uma nova base de dados com o nome de redmine, na segunda linha cria-se um usuário “alexandro” com senha “root”, e finalizando fixa-se todos os privilégios para o usuário alexandro sobre a base de dados redmine.

O próximo passo é copiar o projeto Redmine diretamente do repositório para o computador, na pasta /var/www. Primeiramente insere-se o comando para entrar no diretório desejado: “cd /var/www”, posteriormente copia-se o projeto: “sudo svn checkout http://redmine.rubyforge.org/svn/trunk redmine”.

Com o projeto já copiado, entra-se dentro do diretório redmine com o comando “cd redmine”, posteriormente executa-se o comando “sudo cp config/database.yml.example config/database.yml”. Assim a partir do arquivo de exemplo, que está localizado dentro do diretório config, é gerado um novo arquivo dentro do mesmo diretório com o nome de database.yml para se conectar ao DB. A configuração deste arquivo é mostrada no quadro 15, a seguir.

```
production:
adapter: mysql
database: redmine
host: localhost
username: alexandro
password: root
encoding: utf8

development:
adapter: mysql
database: redmine
host:localhost
username: alexandro
password: root
encoding: utf8
```

Quadro 15: Configuração do arquivo database.yml

Com estas configurações no arquivo database.yml, dois ambientes estarão configurados para se conectar ao DB, o ambiente de produção, e o ambiente de desenvolvimento. O objetivo de configurar o ambiente de desenvolvimento, é facilitar o desenvolvimento dos plugins, desta forma não será necessário reiniciar o servidor.

Deve-se gerar um armazenamento de sessões secreta, com o comando: “rake generate_session_store”.

Feita estas configurações, tudo está preparado para gerar a estrutura do DB, com o comando “rake db:migrate”. Este comando cria todas as tabelas necessárias e o usuário administrador.

Deverão ser inseridos dados de configuração padrão no DB, executando o seguinte comando: “rake redmine:load_default_data”. Fixando pt-BR como linguagem default.

O usuário que executa Redmine deve ter permissão de escrita nos seguintes subdiretórios: files, log e tmp. Assumindo usuário alexandro: “sudo chown -R alexandro:alexandro files log tmp public/plugin_assets” e “sudo chmod -R 755 files log tmp public/plugin_assets”.

Agora a instalação está concluída, basta iniciar o servidor “ruby script/server” e acessar o endereço “http://localhost:3000/”.

7.2 CRIAÇÃO DE PLUGINS

Gerar um plugin para o Redmine é algo muito simples, pode ser usado o próprio gerador do Redmine, como exemplo será criado um plugin para enquetes. Para gerar o plugin deve ser executado o comando: “ruby script/generate redmine_plugin Enquetes”.

O novo plugin com o nome de enquetes já esta criado, seu diretório esta em “vendor/plugins/redmine_enquetes”. A estrutura de diretórios e arquivos é apresentada pela figura 9.

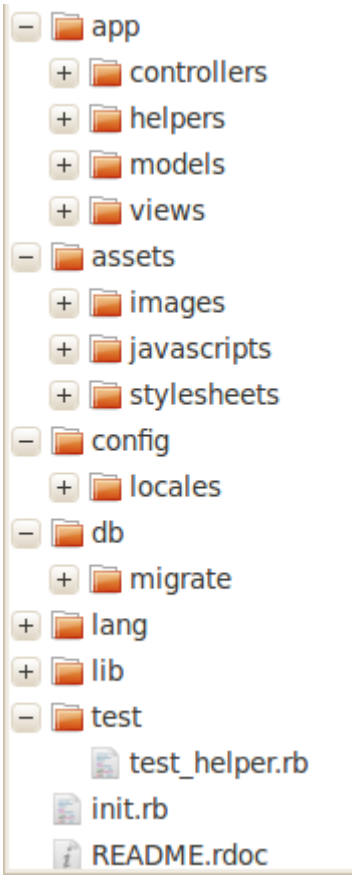


Figura 9: Estrutura de diretórios e arquivos de um plugin Redmine.

A estrutura é semelhante à de um projeto Ruby on Rails, apenas os diretórios vendor, tmp, script, doc e lib não são presentes em um plugin. O diretório assets tem o mesmo propósito do diretório public de um projeto Rails, os objetivos dos demais diretórios também são os mesmos, bastando consultar no item 2.4 deste trabalho.

O próximo passo é ajustar o arquivo de inicialização “redmine_enquetes/init.rb”, para ajustar as informações do plugin referentes a nome, autor, descrição e versão, no plugin representados por name, author, description e version respectivamente, como é mostrado no quadro 16.

```
require 'redmine'

 Redmine::Plugin.register :redmine_enquetes do
  name 'Enquetes plugin'
  author 'Alexandro Luiz Hilleshein'
  description 'Plugin para enquetes'
  version '0.0.1'
 end
```

Quadro 16: Registrando o plugin no arquivo de inicialização.

Com as informações do Quadro 16, o plugin já estará registrado no Redmine. Iniciando a aplicação e acessando http://localhost:3000/admin/plugins, as informações do registro serão apresentas, assim como mostrado pela figura 10.

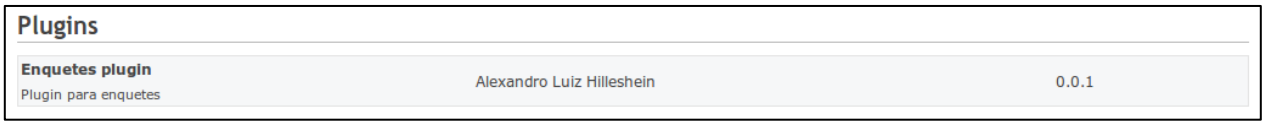


Figura 10: Apresentando informações de registro.

O plugin Enquetes está criado e registrado, porém ainda não faz nada, então o próximo passo e criar um model: “ruby script/generate redmine_plugin_model enquetes enquete questao:string nao:integer sim:integer”.

Foi gerado o model Enquete em “app/models/enquete.rb”, e seu respectivo arquivo de migração em “db/migrate/20111116111312_create_enquetes.rb”. Mas o arquivo de migração deve ser renomeado, pois arquivo de migrações de plugin Redmine, não podem ter prefixados timestamp, este tipo de configuração não é suportado pelo Redmine. O prefixo timestamp das migrações pode ser substituído por “001”, “002”, “003”, etc. O arquivo “20111116111312_create_enquetes.rb” renomeado fica “001_ create_enquetes.rb”.

Agora pode ser migrado para o DB, usando o comando: “rake db:migrate_plugins”.

Com as tabela enquetes criada, é possível adicionar novas questões no banco. Basta entrar no console “ruby script/console”, e executar os comandos do quadro 17.

```
Enquete.create(:questao => "Você pode ver esta questão?")
Enquete.create(:questao => "E esta questão, você pode ver também?")
```

Quadro 17: Inserindo questões no DB.

O comando “create” é do Rails, com as linhas do quadro acima ele cria dois novos objetos Enquete, e grava-os no DB.

O model Enquete “app/models/enquete.rb” deve ser editado, um método com o nome de votar é necessário para posteriormente ser acessado pelo controlador.

```
class Enquete < ActiveRecord::Base
    def votar(resposta)
     increment(resposta == 'sim' ? :sim : :nao)
    end
end
```

Quadro 18: Model Enquete

Falta criar um controller, para existir uma interface de votação, a sintaxe do gerador é: “ruby script/generate redmine_plugin_controller Enquetes enquetes index votar”.

Com a linha acima, é criado o controller Enquetes, dois métodos index e votar, e também as respectivas views para cada método index.html.erb e votar.html.erb.

O arquivo app/controllers/enquetes_controller.rb deve ser editado, ficando como mostrado no quadro 19.

```
class Enquetes_Controller < ApplicationController
  unloadable

  def index
     @enquetes = Enquete.find(:all)
  end

  def votar
     enquete = Enquete.find(params[:id])
     enquete.votar(params[:resposta])
     if enquete.save
      flash[:notice] = 'Voto salvo!'
      redirect_to :action => 'index'
     end
  end
end
```

Quadro 19: Controller Enquetes

Editar também o arquivo "app/views/enquetes/index.html.erb", deixando-o como mostrado no quadro 20.

```
<h2>Enquetes</h2>

<% @enquetes.each do |enquete| %>
  <p>
  <%= enquete[:questao] %>?
  <%= link_to 'Sim', { :action => 'votar', :id => enquete[:id], :resposta => 'sim' }, :method => :post %> (<%= enquete[:sim] %>)
  <%= link_to 'Não', { :action => 'votar', :id => enquete[:id], :resposta => 'nao' }, :method => :post %> (<%= enquete[:nao] %>)
  </p>
<% end %>
```

Quadro 20: View index.html.erb

Também é necessário editar novamente o arquivo "init.rb", adicionando mais uma linha, com mostrado no quadro 21.

```
require 'redmine'

Redmine::Plugin.register :redmine_enquetes do
  name 'Enquetes plugin'
  author 'Alexandro Luiz Hilleshein'
  description 'Plugin para enquetes'
  version '0.0.1'
  menu :application_menu, :enquetes, { :controller => 'enquetes', :action => 'index' }, :caption => 'Enquetes'
end
```

Quadro 21: Arquivo init.rb

O arquivo "view/enquetes/votar.html.erb" pode ser removido.

Está faltando apenas acrescentar um link em algum lugar na interface do Redmine, para acrescentar um link na pagina inicial, o arquivo "init.rb" deve ser novamente editado.

Agora basta acessar o endereço http://localhost:3000/enquetes.

8 MYSQL

MySQL é um sistema de gerenciamento de DB relacional que utiliza a linguagem de consulta estruturada SQL como interface de acesso e extração de informações do DB em uso. O MySQL é um dos sistemas de gerenciamento de DB mais populares e usados no mundo. É muito rápido, multitarefa e multiusuário. (MANZANO,2007)

O MySQL está em crescente uso, pois permite integração com diversas linguagens de programação e atende as necessidades da maioria das aplicações. Além de, segundo Manzano (2007), ser um servidor confiável, rápido e de fácil utilização, que pode ser usado com grandes bancos de dados, considerando aplicações voltadas para a Internet.

9 BANCO DE DADOS RELACIONAL

Um Banco de Dados relacional armazena os dados e informações em tabelas que são formadas por linhas (seus registros) e colunas (seus campos). Atualmente os Sistemas de Gerenciamento de Bancos de Dados (SGBDs), são ferramentas baseadas em Banco de Dados relacional. (MANZANO, 2007)

10 HTML

A HTML (Hipertext Markup Language) é uma linguagem da World Wide Web, que foi desenvolvida no final do anos 80 e no início dos anos 90. Uma página HTML é um arquivo de texto plano que pode ser criado com qualquer processador de texto, onde contém uma série de instruções que orientam o programa browser (navegador de internet), como apresentar página Web. O poder do HTML vem pelo fato de que qualquer computador que rode um programa

browser pode lê-lo e apresentá-lo, não importando se o computador é um PC rodando Windows, um sistema com base Unix, ou um Mac. (BREMMER; IASI; SERVATI, 1998)

A linguagem HTML é usada para estruturar conteúdo em uma página Web. Esta estrutura segue padrões para que os browsers consigam interpretá-la. Uma estrutura básica de um documento HTML é apresentada pelo quadro 22.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pt">
  <head>
    <title>Título do Documento</title>
  </head>
  <body>
    Aqui é escrito o corpo da página
  </body>
</html>
```

Quadro 22: Estrutura básica de uma página HMTL.

11 CSS

CSS (Cascading Style Sheets) é uma linguagem que define estilos de layout de documentos HTML. Onde HTML é usada para estruturar o conteúdo de uma página Web, e CSS é usado para formatar este conteúdo, controlando fontes, cores, imagens de fundo, margens e posicionamentos, inserindo assim meios mais modernos para a projeção de paginas da web. Cascading Style Sheets significa folha de estilo em cascata, permite controlar o layout de várias paginas com uma única folha de estilos.

12 PROCEDIMENTOS METODOLÓGICOS

Este trabalho objetiva desenvolver duas ferramentas, uma que auxilie na medição e estimativa e outra para gerenciamento de requisitos em projetos de software. Contudo, neste tópico serão descritos os procedimentos de criação destas ferramentas.

O Redmine é um excelente e popularizado gerenciador de projetos, o que o torna uma ferramenta adequada para inserir funcionalidades objetivadas por este trabalho. Seria inconveniente instalar uma ferramenta apenas para gestão de requisitos, ou então para medição e estimativa, é muito mais simples apenas adicionar funcionalidades como estas, em uma ferramenta que já está instalada. E ao mesmo tempo centralizando serviços relacionados, afinal tudo faz parte de um projeto de software.

Se tratando de plugins para o Redmine, obrigatoriamente são desenvolvidos na linguagem de programação Ruby juntamente com o framework Rails. As interfaces dos plugins também são construídas semelhantemente à interface do Redmine, para que ela fique homogênea, sem causar estranheza para o usuário.

Abaixo estarão descritas cada etapa que será cumprida para a construção de cada plugin:

Especificação das funcionalidades juntamente com APF: como APF estima tamanho a partir das funcionalidades de um sistema, para não estender muito este trabalho não serão demonstradas as especificações de todos os requisitos, apenas das funcionalidades. Para o Requirements Plugin, serão especificadas as funcionalidades juntamente com a aplicação do método APF, com a finalidade de exemplificar do método.

Modelagem: Esta fase é dedicada para modelagem dos sistemas, a modelagem Entidade Relacionamento e a Modelagem UML. A modelagem UML será resumida em um diagrama de classes, pois se trata de dois projetos simples.

Implementação: com base nos requisitos e na modelagem acontece a implementação das ferramentas.

Como o objetivo deste trabalho é o desenvolvimento de dois plugins, os procedimentos para criação de cada um será descrito separadamente, sendo que primeiramente será descrito os procedimentos do plugin Requirements Plugin, e posteriormente do APF Plugin.

13 REQUIREMENTS PLUGIN

É proposto o desenvolvimento de um plugin para o Redmine a fim de gerenciar requisitos de software. O objetivo do plugin é facilitar o processo de gerenciamento de requisitos, possibilitando um maior controle e organização das informações por parte da gerência do projeto. O plugin deve possibilitar o cadastro, alteração e exclusão dos requisitos adicionais, funcionais e não funcionais. Devem existir telas de listagem dos requisitos funcionais e adicionais por projeto, e também de requisitos não funcionais associados a requisitos funcionais, possibilitando a exportação para formato PDF.

13.1 ESPECIFICAÇÃO DAS FUNCIONALIDADES JUNTAMENTE COM APF

APF mede um software a partir de suas funcionalidades, sendo assim já no início do projeto com a determinação de suas funcionalidades, é possível estimar o tamanho do projeto.

13.1.1 Determinação do Tipo de Contagem

Este é o primeiro passo do processo de contagem de pontos por função, e será determinado o tipo de contagem que será utilizado para medir o software. O tipo de contagem do projeto Requirements Plugin é de um projeto de desenvolvimento. Pois trata-se de uma nova aplicação, sendo assim será construída a contagem das funcionalidades requeridas pelo projeto.

13.1.2 **Identificar o escopo da contagem e fronteira da aplicação**

Segundo Vazquez, Simões e Albert (2005, p. 57), a fronteira da aplicação é a interface conceitual que delimita o software que será medido e o mundo exterior ou usuários. Quando há

mais de uma aplicação incluída no escopo de uma contagem, várias fronteiras devem ser identificadas.

Como se trata de um software simples, ou seja, apenas administrar os requisitos de um projeto de software, trata-se de uma aplicação. O escopo da contagem abrange todas as funcionalidades do sistema.

13.1.3 **Contar funções do tipo dado**

As funções do tipo dados representam as funcionalidades fornecidas pelo sistema ao usuário, para atender a suas necessidades de dados. São classificadas em Arquivo Lógico Interno (ALI), e Arquivo de Interface Externa (AIE). (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 59)

A seguir estão tabeladas as funções do tipo dados. A tabela 10 mostra os Arquivos Lógicos Internos (ALIs), e a tabela 11 os Arquivos de Interface Externa (AIEs).

Tabela 10: ALIs do Requirements Plugin

ALIs		
Descrição da Função	Complexidade	Tamanho(PF)
Requisitos funcionais	baixa	7 PF
Requisitos não funcionais	baixa	7 PF
Requisitos adicionais	baixa	7 PF

Os arquivos Requisitos funcionais, Requisitos não funcionais e Requisitos adicionais, são ALIs, pois todos são controlados pela própria aplicação. Todos são de complexidade baixa, pois são poucos os Tipos de Dados (TD) e apenas um Tipo de Registro (TR).

Tabela 11: AIEs do Requirements Plugin

AIEs		
Descrição da Função	Complexidade	Tamanho(PF)
Usuários	baixa	5 PF
Projetos	baixa	5 PF

Os arquivos Usuários e Projetos são AIEs, pois pertencem a outras aplicações, Controle de Segurança e Projeto respectivamente. Tanto Usuários como Projetos são arquivos de complexidade baixa, pois, embora nestes arquivos existam muitos tipos de dados e registros, poucos serão acessados.

Tabela 12: Contribuição das funções do tipo dado para o projeto Requirements Plugin.

Tipo de Função	Complexidade Funcional	Totais por Tipo de Complexidade.	Totais por Tipo de Função.
ALI	3 Baixa x 7	= 21	
	0 Média x 10	= 0	21
	0 Alta x 15	= 0	
AIE	2 Baixa x 5	= 10	
	0 Média x 7	= 0	10
	0 Alta x 10	= 0	

13.1.4 **Contar funções do tipo transação**

"As funções do tipo transação representam a funcionalidade oferecida ao usuário para atender as suas necessidades de processamento de dados pela aplicação. São classificadas em Entradas Externas (EEs), Saídas Externas (SEs) ou Consultas Externas (CEs)." (VAZQUEZ; SIMÕES; ALBERT, 2005, p. 101)

Tabela 13: EEs em Requirements Plugin

EEs		
Incluir requisito funcional	Baixa	3 PF
Alterar requisito funcional	Baixa	3 PF
Excluir requisito funcional	Baixa	3 PF
Incluir requisito não funcional	Baixa	3 PF
Alterar requisito não funcional	Baixa	3 PF
Excluir requisito não funcional	Baixa	3 PF
Incluir requisito adicional	Baixa	3 PF
Alterar requisito adicional	Baixa	3 PF
Excluir requisito adicional	Baixa	3 PF

Tabela 14: CEs em Requirements Plugin

CEs		
Listar requisitos funcionais	Simples	3PF
Consultar requisitos funcionais	Simples	3PF
Listar requisitos não funcionais	Simples	3PF
Consultar requisito não funcional	Simples	3PF
Listar requisito suplementar	Simples	3PF
Consultar requisito suplementar	Simples	3PF

A tabela 15 mostra a contribuição das funções do tipo transação para o projeto Requirements Plugin.

Tabela 15: Contribuição das funções do tipo transação para Requirements Plugin.

Tipo de Função	Complexidade Funcional	Totais por Tipo de Complexidade.	Totais por Tipo de Função.
EEs	9 Baixa x 3	= 27	
	0 Média x 10	= 0	27
	0 Alta x 15	= 0	
CEs	6 Baixa x 3	= 18	
	0 Média x 7	= 0	18
	0 Alta x 10	= 0	

13.1.5 **Determinar o valor do fator de ajuste**

Como citado na fundamentação teórica deste trabalho, o fator de ajuste é opcional. Para o projeto Requirements Plugin, que é um projeto muito simples, torna-se desnecessário calcular o fator de ajuste.

13.1.6 **Cálculo dos pontos de função ajustados**

Requirements Plugin está em fase de desenvolvimento, a fórmula aplicada é:

DFP = (UFP + CFP) x VAF

Onde:

DFP: Número de pontos de função do projeto de desenvolvimento.

UFP: Número de pontos de função não ajustados das funções disponíveis após a instalação.

CFP: Número de pontos de função não ajustados das funções de conversão.

VAF: Valor do fator de ajuste.

Aplicando então a fórmula do projeto de desenvolvimento, tem-se:

UFP= 100

CFP=0 (primeira instalação)

VAF=1 (sem influência)

DFP= (76 + 0) x 1 = 76 pontos de função ajustados.

13.2 DIAGRAMA DE CLASSES

O diagrama de classes é a base para a construção do Requirements Plugin, por este motivo é demonstrado pela figura 11.

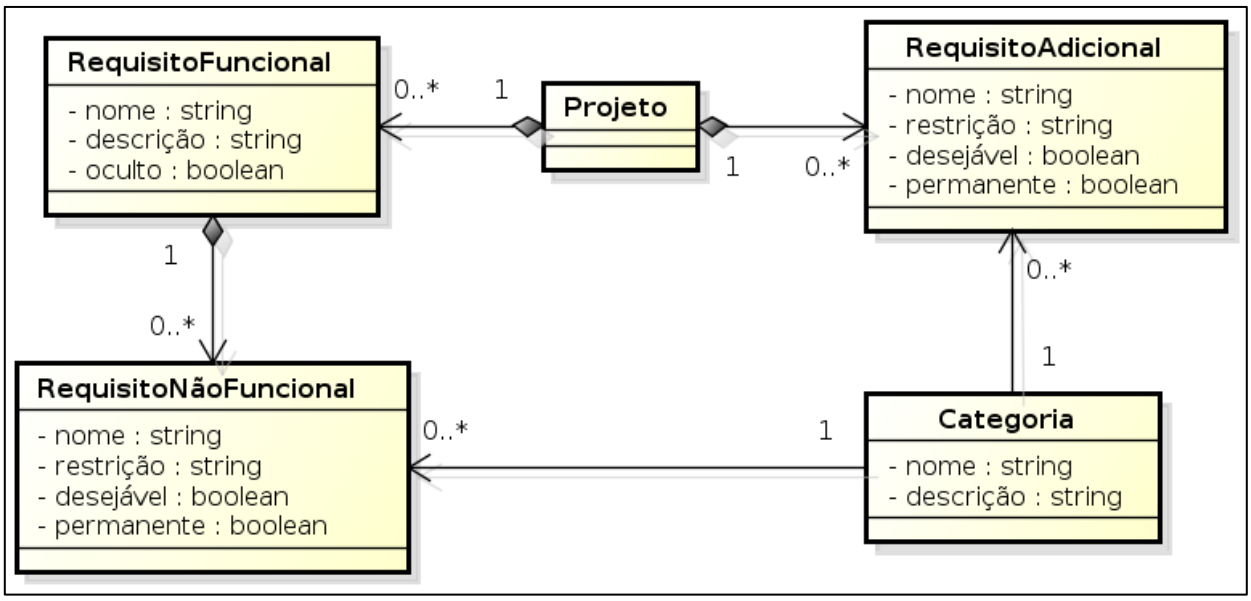


Figura 11: Diagrama de classes do Requiremets Plugin.

13.3 IMPLEMENTAÇÃO

O Requirements Plugin é planejado para ser instalado facilmente, por este motivo o arquivo de migração “db/migrate/001_create.rb”, se encarrega de criar todas as tabelas necessárias para o plugin no DB. Na migração também são criadas as categorias para requisitos adicionais e não funcionais, assim como sugeridas por WAZLAWICK (2004, p. 42).

Toda a interface do plugin é construída semelhantemente à interface do Redmine, a tela de listagem de requisitos funcionais é mostrada no APÊNDICE A deste trabalho.

13.4 INSTALAÇÃO E CONFIGURAÇÃO

O plugin encontra-se disponível para download no endereço https://github.com/AlexandroLuiz/redmine_requirements.

Por ser um plugin nos padrões Rails e não possuindo qualquer dependência com outra ferramenta, a não ser as próprias do Redmine, sua instalação se torna bem trivial. Para instalar o plugin Requirements Plugin, basta colocar o plugin dentro da pasta “vendor/plugins” do projeto Redmine, e via console dentro do diretório do projeto Redmine executar o comando “rake db:migrate_plugins”. Todas as tabelas necessárias serão criadas, adicionando também as categorias para requisitos não funcionais e adicionais.

No Redmine o plugin é interpretado como um módulo, para utilizá-lo em um projeto é preciso marcar o módulo Requirements. Esta configuração do projeto é encontrada em “Configurações >> Módulos”, bastando selecionar o módulo Requirements.

O plugin aceita diferentes permissões, que divididas em permissões para:

- Ver requisitos funcionais;
- Ver requisitos adicionais;
- Ver requisitos não funcionais;
- Criar e editar requisitos funcionais;

- Criar e editar requisitos adicionais;
- Criar e editar requisitos não funcionais;
- Excluir requisitos funcionais;
- Excluir requisitos adicionais;
- Excluir requisitos não funcionais;
- Administrar categorias para requisitos não funcionais;

Estas permissões somente podem ser alteradas por um usuário com perfil de administrador. As configurações de permissões podem ser alteradas em "Administração >> Papéis e permissões", onde podem ser dadas diferentes permissões para cada papel.

13.5 INTEGRAÇÃO COM O REDMINE

Além de ser simples a integração de um plugin com o Redmine, tem uma série de métodos prontos que são fornecidos. Vamos citar alguns destes métodos que foram utilizados no Requirements Plugin. A figura 12 mostra exemplos.

```
1 class FunctionalRequirementsController < ApplicationController
2   unloadable
3
4  layout 'base'
5  menu_item :requirements
6  #filter the current project
7  before_filter :current_project
8  #filtro para analizar as permições
9  before_filter :authorize, :only => [:index, :show, :new, :edit, :destroy, :move ]
10
11   require 'pdf_helper'
12   include PDFHelper
13
14   # GET /functional_requirements
15   # GET /functional_requirements.xml
16   def index
17     @functional_requirements = FunctionalRequirement.find_all_by_project_id(@project.id)
#@project.functional_requirements
```

Figura 12: Classe Controladora Functional Requirements.

Começando pela primeira linha, onde a classe controladora "FunctionalRequirementsController" é uma herança de ApplicationController. No Rails por padrão, todos os controladores herdam da classe ApplicationController, pela razão de que os métodos genéricos para um ou mais controladores são inseridos nesta classe e estão disponíveis para todas as outras. Porém, a classe a qual "FunctionalRequirementsController" está herdando não pertence ao plugin e sim ao Redmine, assim todos os métodos de ApplicationController também serão herdados.

O item destacado no menu (linha5), e as permissões de acesso a métodos (linha 9) são métodos prontos fornecidos pelo Redmine. Assim como o auxiliador para criação de arquivos no formato PDF (linha 11 e 12).

14 APF PLUGIN

É proposto o desenvolvimento de um plugin para o Redmine, que com base na técnica Análise de Pontos por Função (APF) auxilie no processo de medição e estimativa de software. O plugin recebe como entrada funcionalidades de um projeto de software, e a partir de características como, quantidade de Tipos de Registros (TR) juntamente com quantidade de Tipos de Dados (TD) e quantidade de Arquivos Referenciados (AR) juntamente com quantidade de Tipos de Dados (TD), que são atributos para Funções do Tipo Dado e Funções do Tipo Transação respectivamente, tem como saída os Pontos por Função (PF), que é a unidade de medida desta técnica.

Desta forma o plugin deve permitir a inclusão, alteração e exclusão de funções do tipo dado e também do tipo transação, juntamente com seus atributos que permitem o cálculo de PF não ajustados. O plugin deve permitir também a configuração das 14 características gerais do projeto, permitindo assim o cálculo dos PF ajustados.

14.1 DIAGRAMA DE CLASSES

O diagrama de classes é a base para o desenvolvimento do plugin, nele podemos observar as relações entre os objetos, ele é exibido pela figura 13, a seguir.

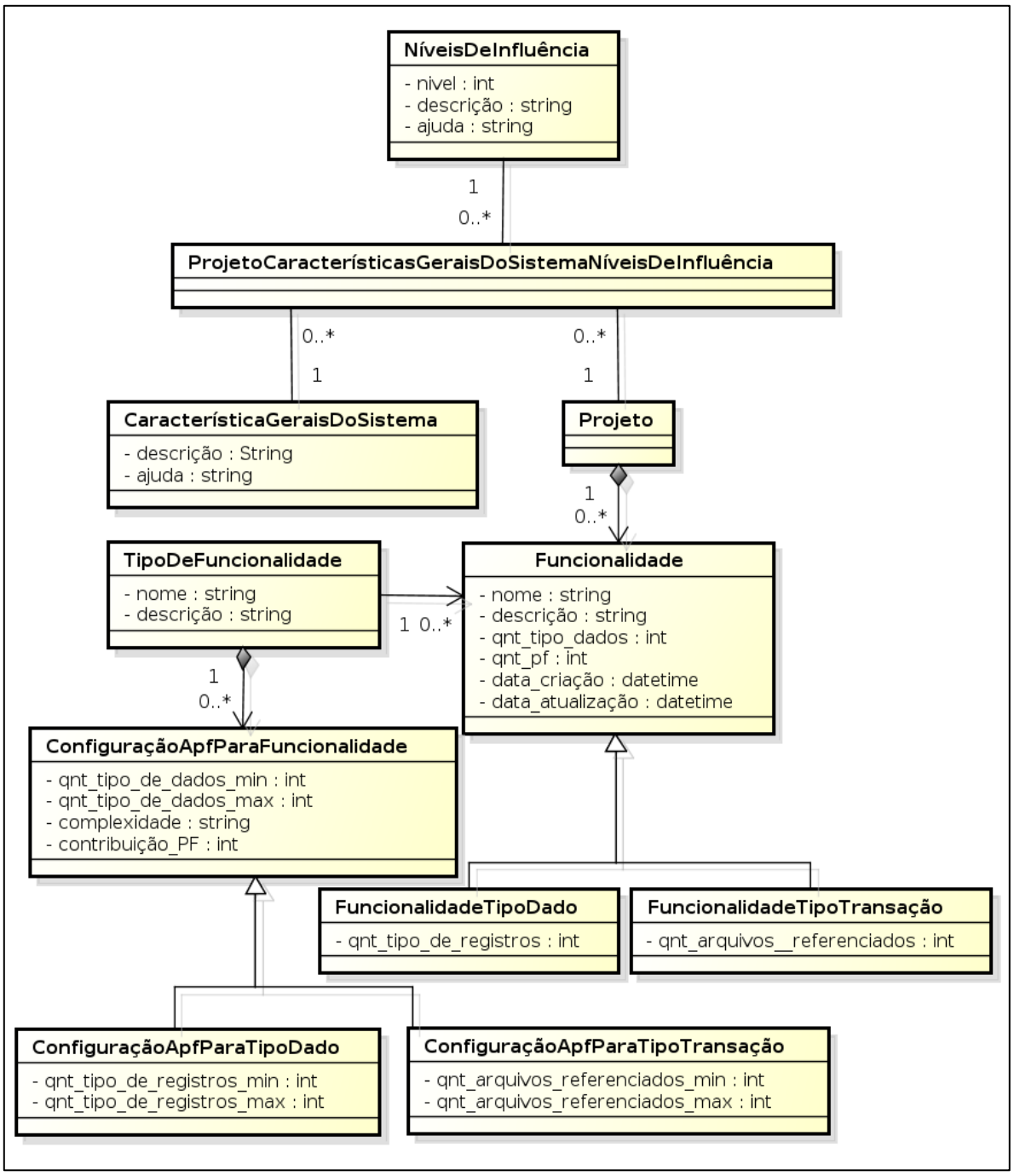


Figura 13: Diagrama de Classes APF Plugin

Algo interessante sobre o diagrama de classes que deve ser enfatizado é o relacionamento entre as classes Projeto e Características Gerais do Sistema, que deveria se estabelecer através da classe Nível de Influência, porém, como os diferentes níveis são estáticos e predeterminados, optou-se por criar uma terceira classe somente para selar esta relação.

14.2 IMPLEMENTAÇÃO

Assim como o Requirements Plugin o APF Plugin também é planejado para ser instalado facilmente, por este motivo os arquivos de migração “db/migrate/009_create.rb” e “db/migrate/004_create_general_caracteristic.rb”, se encarregam de criar todas as tabelas necessárias para o plugin no DB. Na migração também são inseridos no DB os tipos de funcionalidades, as configurações da APF padrão para cálculo dos PF não ajustados, as 14 características gerais do sistema e por fim os 6 níveis de influência predeterminados pela técnica.

A interface do APF Plugin também esta em conformidade com a interface do Redmine. No APÊNDICE B deste trabalho é demonstrada a tela de uma nova funcionalidade do tipo dado.

14.2.1 **Principais métodos**

Os métodos responsáveis pelos cálculos de Pontos por Função (PFs) encontram-se no arquivo “app/helpers/global_helper.rb”. Na figura 14 é demonstrado o método “total_pf”, um dos principais responsáveis pelo cálculo dos PFs.

```
15  def total_pf
16    acum = 0
17     functionality_data_types = FunctionalityDataType.find_all_by_project_id(:project_id)
18     functionality_data_types.each do |functionality_data_type|
19     acum += qnt_pf_by_functionality(functionality_data_type.type_of_functionality.name,
functionality_data_type.qnt_type_register, functionality_data_type.qnt_type_data)
20     end
21
22     functionality_transaction_types = FunctionalityTransactionType.find_all_by_project_id(:project_id)
23     functionality_transaction_types.each do |functionality_transaction_type|
24      acum += qnt_pf_by_functionality(functionality_transaction_type.type_of_functionality.name,
functionality_transaction_type.qnt_reference_file, functionality_transaction_type.qnt_type_data)
25    end
26     acum
27  end
```

Figura 14: Método total_pfs.

Sua função consiste basicamente em recuperar do DB todas as funcionalidades tanto do tipo dados como do tipo transação, posteriormente percorrê-las, invocando o método "qnt_pf_by_functionality" (linhas 19 e 24), passando por parâmetro o nome do tipo da funcionalidade, a quantidade de tipos de registros e a quantidade de tipo de dados, para funções do tipo dado (linha 19), e a quantidade de arquivos referenciados e a quantidade de tipos de dados para funcionalidades do tipo transação (linha 24), acumulando os retornos deste método.

A figura 15 demonstra o método "qnt_pf_by_functionality".

```
29  def qnt_pf_by_functionality(type_of_functionality, qnt_type_register_or_file_references, qnt_type_data)
30    retorno = 0
31    case type_of_functionality
32     when "ALI"
33       configuration_apf_for_data_types = ConfigurationApfForDataType.find_all_by_type_of_functionality_id(1)
34       configuration_apf_for_data_types.each do |conf|
35        if(((qnt_type_register_or_file_references >= conf.type_of_register_min) &&
   (qnt_type_register_or_file_references <= conf.type_of_register_max)) && ((qnt_type_data >= conf.type_of_data_min) &&
   (qnt_type_data <=conf.type_of_data_max)))
36           retorno = conf.pf
37        end
38       end
39     when "AIE"
40        configuration_apf_for_data_types = ConfigurationApfForDataType.find_all_by_type_of_functionality_id(2)
41       configuration_apf_for_data_types.each do |conf|
42        if(((qnt_type_register_or_file_references >= conf.type_of_register_min) &&
   (qnt_type_register_or_file_references <= conf.type_of_register_max)) && ((qnt_type_data >= conf.type_of_data_min) &&
   (qnt_type_data <=conf.type_of_data_max)))
43           retorno = conf.pf
```

Figura 15: Método quantidade de PFs por funcionalidade.

O método "qnt_pf_by_functionality" é responsável por reconhecer o tipo de funcionalidade, consultando no DB as configurações para cada tipo, e retornando os PFs adequados.

14.3 INSTALAÇÃO E CONFIGURAÇÃO

O plugin encontra-se disponível para download no endereço eletrônico https://github.com/AlexandroLuiz/redmine_apf.

Por ser um plugin nos padrões Rails e não possuindo qualquer dependência com outra ferramenta, a não ser as próprias do Redmine, sua instalação se torna bem trivial. Para instalar o plugin APF Plugin, basta colocar o plugin dentro da pasta "vendor/plugins" do projeto Redmine, e via console dentro do diretório do projeto Redmine executar o comando "rake db:migrate_plugins". Todas as tabelas e demais configurações necessárias para perfeito funcionamento do plugin serão criadas.

Agora, para começar a utilizar o plugin no Redmine basta marcar o módulo APF. Esta configuração do projeto é encontrada em "Configurações >> Módulos", bastando selecionar o módulo APF.

O plugin aceita diferentes permissões, que divididas em permissões para:

- Ver funcionalidades do Tipo Dados;
- Ver funcionalidades do Tipo Transação;
- Ver Configurações de Fator de Ajuste;
- Criar e Atualizar Funcionalidades do Tipo Dado;
- Criar e Atualizar Funcionalidades do Tipo Transação;
- Editar Configurações APF para funcionalidades do Tipo Dado;
- Editar Configurações APF para funcionalidades do Tipo Transação;
- Ver Configurações APF para funcionalidades do Tipo Dado;
- Ver Configurações APF para funcionalidades do Tipo Transação;
- Excluir funcionalidades do Tipo Dados;
- Excluir funcionalidades do Tipo Transação;
- Configurar Fator de Ajuste;

Estas permissões somente podem ser alteradas por um usuário com perfil de administrador. As configurações de permissões podem ser alteradas em "Administração >> Papéis e permissões", onde podem ser dadas diferentes permissões para cada papel.

15 ASSOCIAÇÃO ENTRE OS PLUGINS REQUIREMENTS PLUGIN E APF PLUGIN

Fisicamente, ou então via código fonte não existe nenhuma associação entre os dois plugins, porém, na prática os dois plugins estão intimamente relacionados. Na verdade o plugin APF é dependente do plugin Requirements, pois assim como já fundamentado neste trabalho a técnica Análise de Pontos por Função requer os requisitos de software para estimar ou medir o tamanho funcional do projeto.

Utilizar o Requirements Plugin isoladamente de certa forma faz sentido, pois com ele conseguimos gerenciar os requisitos de um projeto de software, posteriormente podemos usufruir destes para gerar tarefas no próprio Redmine. Porém, o plugin Requirements agrega uma função bastante interessante quando unido ao APF Plugin, as funcionalidades inseridas no APF devem ser geradas com base nos requisitos dispostos no Requirements, assim como as características gerais do sistema devem ser configuradas de acordo com os requisitos adicionais.

16 CONSIDERAÇÕES FINAIS

O trabalho apresentou dois estudos, que estão intimamente relacionados no universo da Engenharia de Software. Um primeiro sobre requisitos de software, e um segundo sobre a técnica APF, aplicando estes dois estudos no desenvolvimento de plugins para o Redmine com denominação de Requirements Plugin e APF Plugin respectivamente.

Como revelado no parágrafo anterior, os dois objetos de estudo deste trabalho estão intimamente relacionados em um projeto de software. Não é possível conhecer a técnica APF, sem antes conhecer requisitos de software, ou melhor, só faz sentido aplicar a técnica APF se os requisitos de software forem corretamente levantados. Com este raciocínio percebesse que as duas ferramentas desenvolvidas devem ser usadas conjuntamente.

Com o Requirements Plugin os interessados em um projeto de software, terão uma visão clara dos requisitos e restrições do projeto, levando em consideração que o mesmo seja bem planejado. Outras vantagens são a centralidade, acessibilidade e organização no que diz respeito aos requisitos e ao projeto de software como um todo.

O APF Plugin torna-se uma ferramenta muito útil assim que os requisitos de software forem revelados, bastando observar os requisitos para construção das funcionalidades e configuração das características gerais do projeto.

Deve-se enfatizar que as ferramentas desenvolvidas neste trabalho, possuem licença GPL 2.0, desta forma qualquer pessoa ou organização tem total liberdade para fazer edição ou uso das mesmas.

Contudo, o desenvolvimento deste trabalho além da agregação de conhecimento, trás enorme satisfação pela contribuição ao software livre.

17 REFERÊNCIAS

AKITA. Fabio. **Repensando a Web com Rails.** Rio de Janeiro: Brasport, 2006.

BREMMER, Lynn M.; IASI, Antony F.; SERVATI, Al. **A Bíblia da Intranet:** Tudo o que você precisa aprender sobre Intranets. São Paulo: Makron Books, 1998.

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. **Análise Orientada a Objetos.** Florianópolis: Visual Books, 2001.

DAVID, Marcio Frayze. Programação Orientada a Objetos: uma introdução. **Guia do Hardware,** outubro de 2007. Disponível em: <http://www.hardware.com.br/artigos/programacao-orientada-objetos/>. Acesso em: 26 maio 2011.

FERNANDEZ, Obie et al. **Programando Rails**: "A Bíblia". Rio de Janeiro: Alta Books, 2008.

FLANAGAN, David; MATSUMOTO, Yukihiro. **A Linguagem de Programação Ruby**. Rio de Janeiro: Alta Books, 2008. 354 p.

FOWLER, Martin. **UML Essencial:** Um breve guia para a linguagem-padrão de modelagem de projetos. 3a. ed. . Porto Alegre: Bookman, 2005.

LISBOA, Flávio Gomes da Silva. **Zend Framework:** Desenvolvimento em PHP 5 orientado a objetos com MVC. São Paulo: Novatec, 2008.

PAULA FILHO, Wilson de Páduo. **Engenharia de software:** fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2009. 1248 p.

PENDER, Tom. **UML:** a Bíblia. Rio de Janeiro: Elsevier, 2004.

PRESSMAN, Roger, **Engelharia de Software**. São Paulo: Pearson Makron Books, 2009.

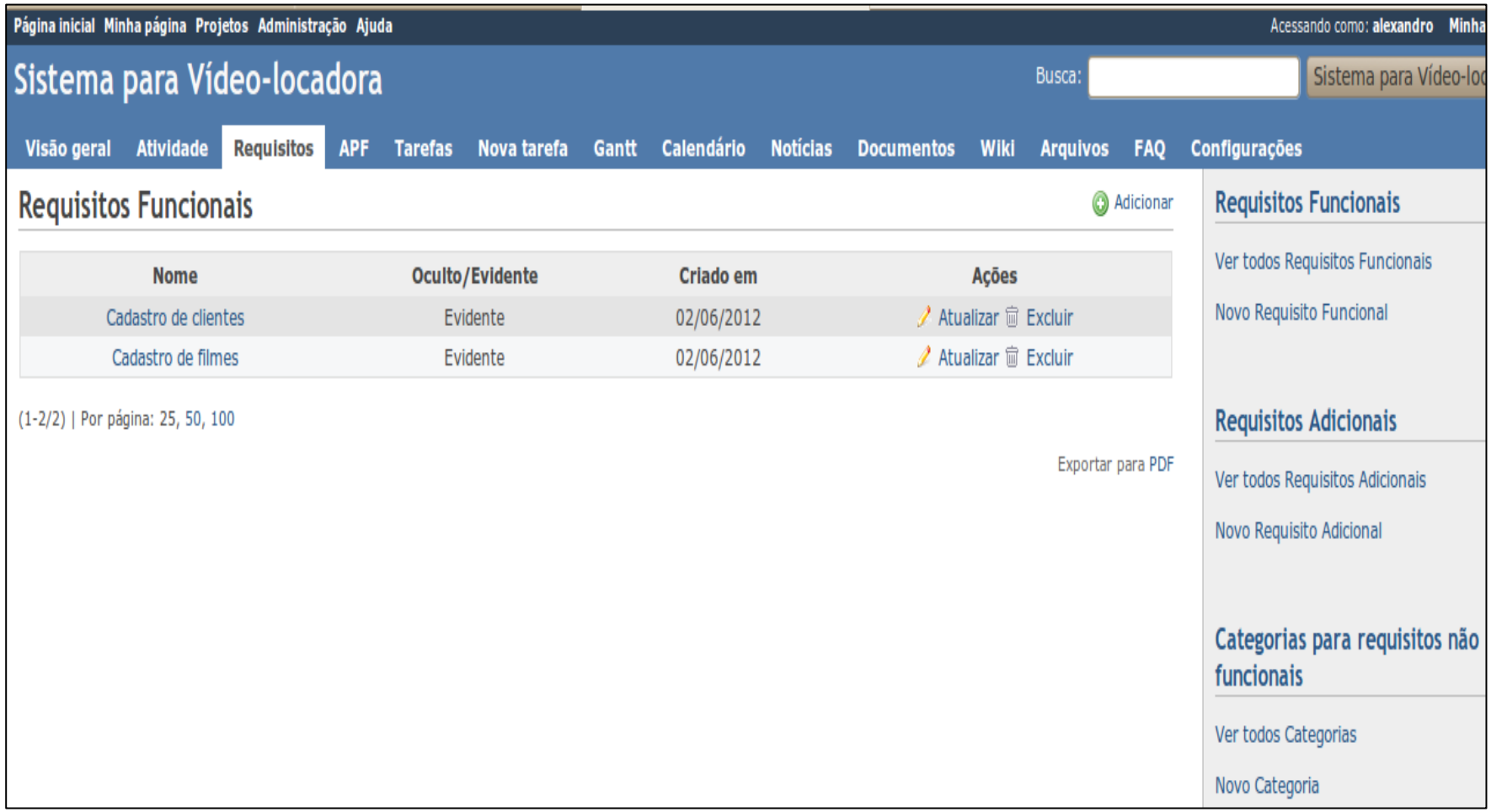
MANZANO, José Augusto. **MySQL 5 interativo:** Guia Básico de Orientação e Desenvolvimento.1 ed. São Paulo: Erica, 2007. 332p.

TATE, Bruce A.; HIBBS, Curt. **Ruby on Rails:** Executando. Rio de Janeiro: Alta Books, 2006.

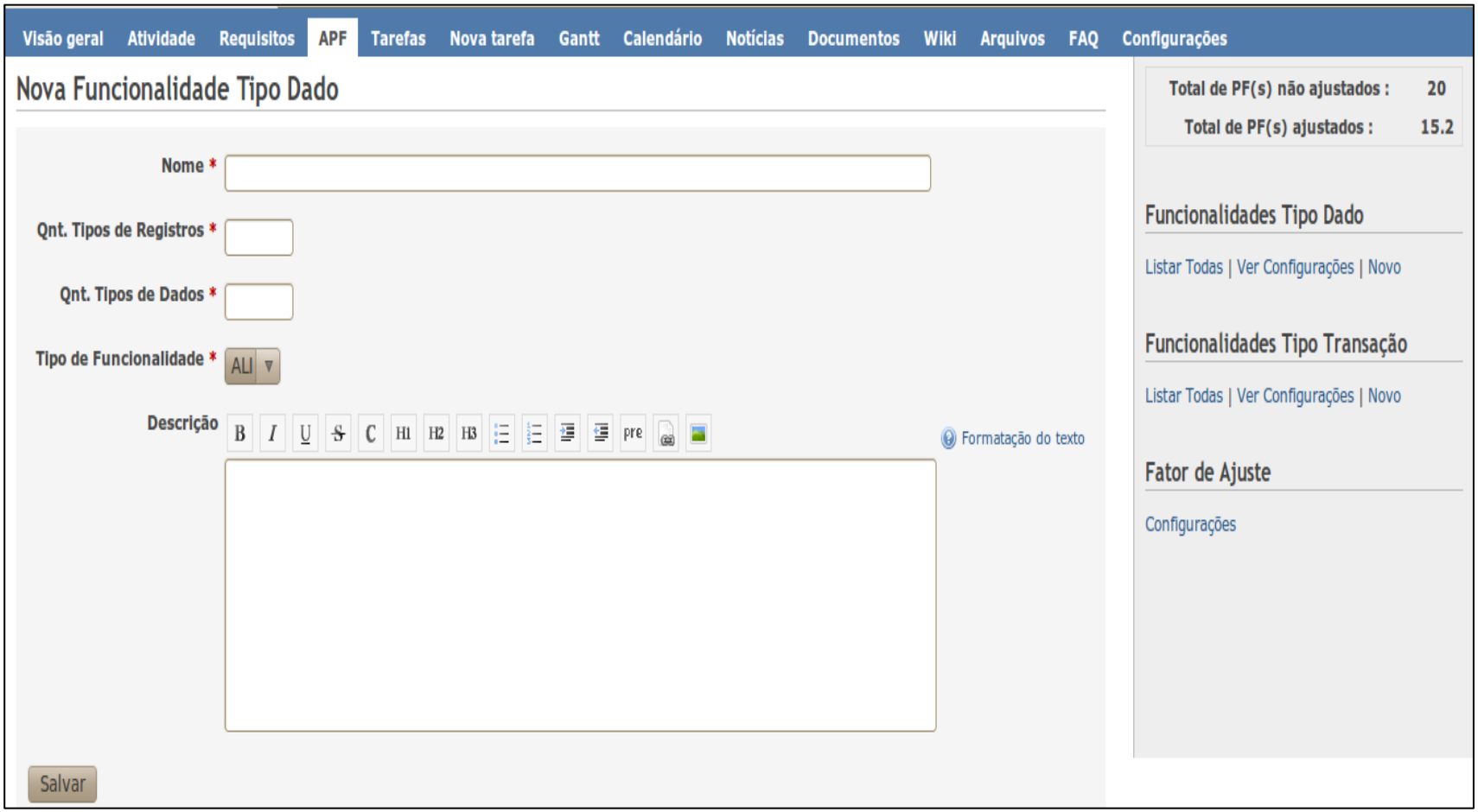
THOMAS, Dave; HANSSON, David Heinemeier. Desenvolvimento Web ágil com Rails. 2. ed. Porto Alegre: Bookman, 2008.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **Análise de Pontos de Função:** Medição, Estimativas e Gerenciamento de Projetos de Software. 4. ed. São Paulo: Editora Érica Ltda, 2005. 230 p.

APÊNDICE A – REQUIREMENTS PLUGIN



APÊNDICE B – APF PLUGIN



APÊNDICE C – APF PLUGIN