A Utilização de Engenharia de Domínio para Descoberta de Padrões de Análise em Sistemas de Controle de Mudanças

Ingrid Sathler Corrêa de Lima, Evaldo de Oliveira da Silva

Centro de Ensino Superior de Juiz de Fora – Juiz de Fora – MG – Brasil

{ingridsathler, evaldo.oliveira} @gmail.com

Abstract. *This article attempts through techniques such as reuse, domain engineering, domain analysis describing and cataloging best practices and conceptual standards for future implementations of change control systems , with the purpose of facilitating , save time and money for building systems of the same domain.*

Resumo. *Este artigo busca através de técnicas como o reuso, engenharia de domínio, análise de domínio descrever e catalogar boas práticas e padrões conceituais, para futuras implementações de sistemas de controle de mudanças, com o propósito de facilitar, reduzir tempo e custos para a construção de sistemas de um mesmo domínio.*

1. Introdução

A reutilização de software é um recurso da engenharia de software que busca o aumento da produtividade, qualidade e redução de tempo em desenvolvimento por meio da utilização de todas as informações já adquiridas anteriormente.

Várias técnicas podem ser utilizadas para alcance do reuso, como por exemplo, uso de padrões de software, paradigma do desenvolvimento baseado em componentes, linhas de produto de software e a engenharia de domínio. Para fins deste estudo, foi utilizada a engenharia de domínio, tendo em vista que este trabalho tem o intuito de encontrar esquemas conceituais e posteriormente transformá-los em padrões de análise relevantes para um domínio através de uma análise das ferramentas já existentes.

Padrões de análise são esquemas conceituais de uma parte da aplicação que modelam uma característica ou uma funcionalidade útil a outras aplicações de um mesmo domínio e, portanto, tornam-se importante recurso de reutilização [FOWLER, 1997].

A dificuldade presente neste recurso de reutilização está em entender o domínio da aplicação, identificar esses padrões e posteriormente descrever essas soluções de forma clara e simples, sem redundâncias por meio da engenharia de domínio.

Sabendo-se que a construção de novas aplicações gera custos de investimento e envolvimento elevados, esta pesquisa busca contribuir para a otimização e agilidade nas atividades do processo de desenvolvimento de aplicações. Este trabalho propõe como estudo de caso a utilização de ferramentas de controle de mudanças para a descoberta de padrões de análise que possam contribuir não somente para o entendimento do domínio

destas aplicações, mas também para permitir o reuso de esquemas conceituais para a criação das mesmas.

A escolha desse domínio se deu por ser uma ferramenta frequentemente estudada academicamente e de comum uso nos ambientes profissionais. Outro aspecto relevante foi a facilidade de encontrar ferramentas com esse propósito e a pouca variação que a sua estrutura apresenta.

Esquemas foram identificados e descritos de forma conceitual, garantindo maior conhecimento sobre o domínio deste tipo aplicação. Com base nos resultados encontrados, acredita-se que será possível entender melhor como funcionam as ferramentas de controle de mudanças, e em caso do desenvolvimento de uma nova aplicação, haverá poupança de tempo e gastos desnecessários com estruturas previamente catalogadas.

Este artigo está organizado da seguinte forma: A seção 1. faz uma breve introdução. A seção 2. cita os trabalhos relacionados a reutilização de softwares. A seção 3. faz uma breve introdução sobre padrões de software e seus tipos. A seção 3.1. explora os padrões de análise. A seção 4. fala sobre engenharia de domínio, a subseção 4.1. sobre análise de domínio, a seção 4.2. projeto de domínio. A seção 5 faz uma breve descrição sobre os sistemas de controle de mudanças, onde as ferramentas Mantis, Redmine e TRAC são abordadas em nas subseções. A seção 6 traz a aplicação da engenharia de domínio e a descrição dos esquemas alcançados neste trabalho. Por último a seção 7. traz as considerações finais e sugestões para futuros trabalhos seguido da seção 8. que traz as referências usadas.

2. Reutilização de Software

Com o surgimento e crescimento do uso das linguagens de programação consideradas de alto nível, como por exemplo, Java, PHP, C/C++, a ideia de reutilização tem sido amplamente adotada visando aumentar a qualidade e produtividade durante a implementação de software.

Observa-se atualmente que o desenvolvimento de software vem se deparando com diversos desafios, os quais se podem citar a diminuição de custos e tempo de entrega de produtos no mercado. Estes desafios estão associados à crescente complexidade e aumento do tamanho dos softwares produzidos. Com o propósito de atender a esta realidade, várias abordagens de reutilização de software foram propostas, tais como o uso de padrões de software [GAMMA et al. 1995], *frameworks* [JOHNSON, 1992], o paradigma do desenvolvimento baseado em componentes [ATKINSON et al., 2001], engenharia de domínio [NEIGHBORS, 1989], e linhas de produto de software [CLEMENTS e NORTHROP, 2001].

3. Padrões de Software

De acordo com Fowler (1997, p. 8), um padrão é definido como: “uma ideia que tem sido utilizada em um contexto prático e provavelmente será útil em outros contextos”.

O reuso por meio de padrões teve origem nos estudos de Christopher Alexander, professor da Universidade da Califórnia, localizada em Berkeley nos Estados Unidos.

Christopher Alexander desenvolveu várias teorias sobre padrões de projeto em arquitetura, que foram publicados e catalogados [FOWLER, 1997].

De acordo com Silva (2008), as publicações e especificações elaboradas pelo professor Alexander serviram como referência para os pesquisadores e profissionais da Ciência da Computação, no desenvolvimento de métodos e técnicas sobre a reutilização de padrões de software (*software patterns*). Para ele, padrões facilitam a comunicação entre projetistas por meio de um vocabulário, permitindo que as equipes de desenvolvimento compartilhem modelos, interfaces e códigos. Possibilitam ainda, que a documentação do software seja mais simplificada, pois os projetistas podem levar menos tempo modelando ou escrevendo códigos.

Os padrões de software abrangem diferentes níveis de abstração, sendo classificados em diversas classes, de modo a facilitar sua recuperação e uso. Dependendo da fase ou nível da construção de um sistema de informação, Devedzic (2002), classificou os padrões de software como na tabela abaixo:

Tabela 1. Classificação dos padrões [Devedzic, 2002].

TIPOS DE PADRÕES	
Padrões de Processo	Definem soluções para os problemas encontrados nos processos de desenvolvimento, controle de configuração e testes envolvidos na engenharia de software;
Padrões Arquiteturais	Expressam o esquema ou organização estrutural fundamental de sistemas de software ou hardware;
Padrões de Análise	Descrevem soluções para problemas de análise de sistemas, embutindo conhecimento sobre o domínio de aplicação específico;
Padrões de Projeto	Definem soluções para problemas de projeto de software;
Padrões de Interface	Definem soluções para problemas comuns no projeto da interface de sistemas;
Padrões de Programação	Descrevem soluções de programação particulares de uma determinada linguagem ou regras gerais de estilo de programação;
Padrões de Persistência	Descrevem soluções para problemas de armazenamento em arquivos ou bancos de dados;
Anti-Padrões	Descrevem uma solução ruim para um problema que resultou em uma situação ruim. Além disso, descrevem como escapar de uma situação ruim e como proceder para, a partir dela, atingir uma boa solução.

3.1. Padrões de Análise

Padrões de análise são modelos conceituais de uma parte da aplicação que visam ser reutilizados em outras aplicações de mesmo domínio. Segundo Fowler (1997), um padrão de análise é um mecanismo de reutilização que permite a um projetista menos experiente reutilizar o conhecimento de outros especialistas. Para isso, um padrão de análise descreve um conjunto de classes, possivelmente pertencentes a diferentes hierarquias de classes, e associações existentes entre elas.

Robertson e Strunch (1993) definem um padrão de análise como qualquer parte de uma especificação de requisitos que origina em um projeto e pode ser reusada em um ou em diversos projetos.

Segundo Monteiro (2002), todo padrão especificado deve conter os seguintes itens:

- Nome: usado para identificar o padrão.
- Contexto: descreve o contexto ao qual o padrão se aplica.
- Problema: fornece uma declaração sucinta do problema a ser resolvido.
- Solução: descreve uma solução proposta ou um número de possíveis soluções para o problema. Uma solução pode ser dada de forma narrativa.
- Discussão: tópico que descreve as forças ou requisitos atendidos pelo padrão, as restrições e consequências de uso e as relações com outros padrões.
- Exemplos: devem ser dados exemplos de situações reais.

O interessante é além das técnicas para descobrir e aplicar padrões de análise seria especificá-los e catalogá-los a fim de se ter o controle sobre os padrões existentes, facilitando ainda mais a reutilização dos mesmos.

Catálogo de padrões é uma coleção de padrões, que já foram utilizados e então catalogados de forma a facilitar a reutilização no desenvolvimento de novas aplicações. Um exemplo de catálogo de padrões é o Repositório de Padrões de Projeto de Aplicações de Hipermídia (*Hypermedia Design Patterns Repository*, 2006) que contém uma série de padrões aplicáveis ao desenvolvimento de aplicações de hipermídia. Em seus respectivos livros Gamma (1994) e Fowler (1997) também listam uma série de padrões propostos por eles.

O uso de padrões de análise reduz o tempo de desenvolvimento de novas aplicações, uma vez que antes de começar o desenvolvimento o projetista deve consultar o catálogo de padrões, poupando esforços para encontrar soluções já descritas. Além do tempo, o uso de padrões acaba por reduzir também os custos do projeto, levando em consideração a base de conhecimento prévio já existente.

4. Engenharia de Domínio

O termo “domínio” na engenharia de software pode ser definido como uma coleção de aplicações de softwares que possuem características em comum ou similar. Logo, a engenharia de domínio busca identificar, construir, catalogar e disseminar um conjunto de componentes de software que tenham aplicabilidade em outros softwares dentro de

um domínio de aplicação específico. Seu objetivo é estabelecer um esquema em que engenheiros de software possam compartilhar estes componentes e usá-los em sistemas futuros.

Para Prieto-Diaz e Arango (1991) a engenharia de domínio se preocupa com a identificação e modelagem de características comuns e variáveis em aplicações de um dado domínio, gerando como resultado modelos de domínio (e.g. casos de uso de domínio, classes do domínio) que podem ser reutilizados em aplicações específicas.

A engenharia de domínio compreende algumas etapas muito importantes.

- Definição do domínio, onde determina a família de aplicações que a ser estudada.
- Análise de viabilidade que consiste em levantar estimativas de custo e retorno ao longo do tempo de uma família de aplicações;
- Análise de domínio que consiste em entender e registrar os conceitos gerais de um domínio; e
- Projeto do domínio que visa construir um modelo genérico do domínio para posterior reutilização.

A engenharia de domínio considera que a maioria dos sistemas de software são variações de sistemas que já foram construídos dentro de um mesmo domínio de aplicação. Isto pode ser compreendido para melhorar a qualidade do produto e a produtividade da equipe envolvida, assim como reduzir os custos no longo prazo (FRAKES e KANG, 2005).

A Figura 1 a seguir apresenta as etapas do processo de engenharia de domínio. As próximas seções descrevem cada etapa deste processo.

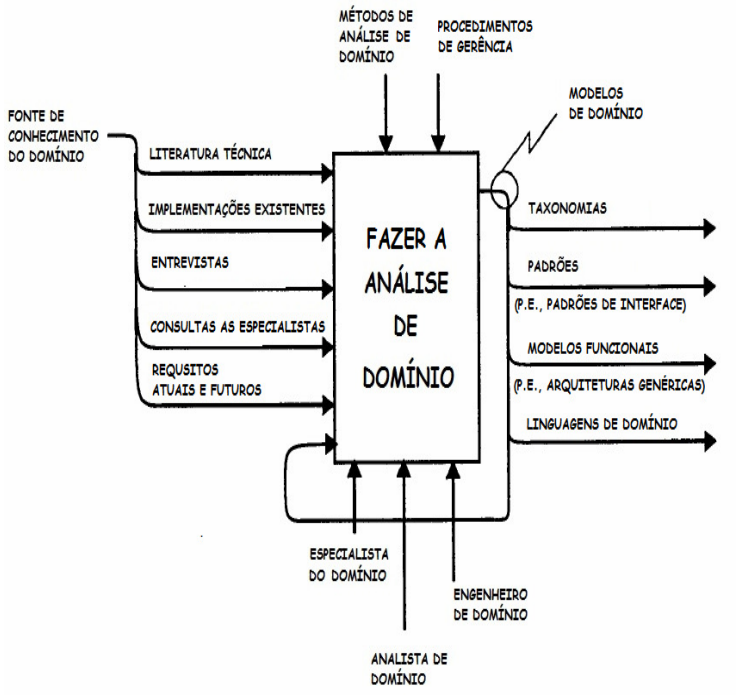

Figura 1. Diagrama de contexto para análise de domínio (Prieto-Diaz 1991)

A informação é recolhida a partir de sistemas existentes, na forma de código-fonte, documentação, projetos, manuais, requisitos e consultas a especialistas do domínio. Os analistas de domínio extraem a informação relevante e junto com o engenheiro de domínio organizam e encapsulam em forma de modelos de domínio, padrões e coleção de códigos e ou classes reutilizáveis. Os modelos de domínio podem ser criados utilizando os diagramas de classe previstos na UML.

O processo de análise de domínio faz parte de um ambiente de desenvolvimento de software e é conduzido por meio de técnicas de gestão, métodos e técnicas de análise de domínio. O papel central é desempenhado pelo analista de domínio que coordena todo processo. O especialista em domínio e engenheiro de domínio tem funções de facilitar as fases de entrada ou aquisição e saída ou encapsulamento de conhecimento.

Este processo está em constante aperfeiçoamento, uma vez que os recursos reutilizáveis são disponibilizados e novos sistemas são construídos, os modelos de domínios são refinados.

O processo de engenharia de domínio também tem o objetivo de estabelecer Linhas de Produto de Software (ou LPS), permitindo que a reutilização seja trabalhada tanto no nível conceitual quanto no nível da arquitetura e dos componentes do software (CLEMENTS e NORTHROP, 2001).

4.1. Análise de Domínio

Segundo Pietro-Diaz (1997), a análise de domínio é associada com a reutilização e a sua finalidade é capturar informações envolvidas com o domínio a ser reutilizado no desenvolvimento de outras aplicações de um mesmo domínio. Então, é possível definir a análise de domínio como o processo em que se identifica, captura e organiza a informação usada no desenvolvimento de software de forma que ela possa ser reutilizada no processo de criação de novos sistemas.

Modelos de domínio ou soluções generalistas para a resolução de um problema que podem ser utilizadas em contextos parecidos é o resultado de um processo de análise de domínio.

4.2. Projeto de Domínio

Ao conjunto das atividades de especificação e implementação da Infraestrutura é geralmente dado o nome de Projeto de Domínio. O principal produto dessa fase é um *framework* reutilizável, que é posteriormente especializado na fase de Projeto da Aplicação Específica, para concretizar uma nova aplicação, cujos requisitos foram especificados numa atividade de Análise de Requisitos da Aplicação Específica (Cima & Werner, 1997).

Para que um *framework* represente as características do domínio de aplicação para o qual foi construído, é necessário que essas características sejam capturadas em um modelo do domínio através da realização da fase de análise de domínio. Desta forma, na fase de projeto de domínio a ênfase é a construção dessa infraestrutura de reuso e duas atividades principais merecem destaque: especificação da infraestrutura e projeto/implementação (Arango & Pietro-Diaz, 1994).

Nessa atividade é onde são confrontados os artefatos gerados na etapa de análise de domínio, buscando obter as especificação das classes, mas ainda sem se preocupar com a fase de implementação. O objetivo é desenvolver uma arquitetura genérica para o domínio e promover sua posterior reutilização na construção de outros sistemas.

A figura 2 ilustra um modelo genérico de desenvolvimento para/com reuso:

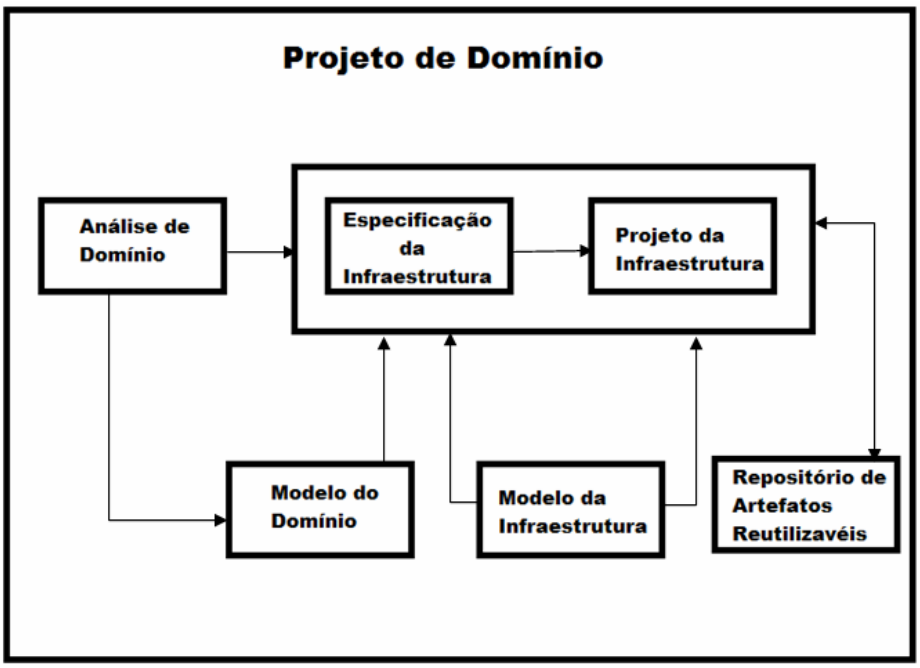

Figura 2. Modelo genérico de desenvolvimento para/com reuso.

5. Sistemas de Controle de Mudanças

Durante todo o ciclo de vida de um software surgem inúmeras solicitações de mudanças tanto de escopo quanto de requisitos, o que causam alguns transtornos ao projeto e aos envolvidos.

As aplicações de controle de mudanças têm como objetivo registrar e gerenciar todas essas mudanças recorrentes ao longo do processo de desenvolvimento e manutenção do software. Por meio dessas ferramentas é possível acompanhar a evolução do projeto e documentá-lo.

Existem várias aplicações com este objetivo no mercado. Entre todas, para análise, utilizaremos as ferramentas *"open-source"* Redmine, TRAC e Mantis. A escolha destas ferramentas foi baseada na facilidade e praticidade que a interface web traz para o usuário, além de serem multiplataformas, o que permite que o sistema seja acessado/executado a partir de vários sistemas operacionais.

5.1. Mantis

O Mantis é uma ferramenta desenvolvida em PHP, disponibilizada gratuitamente desde 2000. É possível acessa-la a partir de qualquer sistema operacional que suporte a linguagem PHP, o servidor Apache e um bancos de dados, que pode ser o MySQL, MS-

SQL Server, PostgreSQL ou DB2. A ferramenta ainda oferece integração com o sistema de controle de versão SVN [http://www.mantisbt.org/.].

Assim como as outras duas ferramentas analisadas neste trabalho, o Mantis é um sistema de controle de mudanças/solicitações, que permite controlar as alterações de vários projetos através dos casos que são relatados. Todo projeto deve ter nome, estado, visibilidade. É preciso também definir se o projeto herda ou não as variáveis globais. O primeiro passo para relatar um novo caso é definir a qual projeto ele pertence, depois temos vários atributos, porém alguns deles são considerados indispensáveis:

- Categoria: relata que tipo de ocorrência está acontecendo, se trata de uma correção, uma funcionalidade ou suporte, por exemplo.
- Frequência: Com qual frequência ocorre
- Gravidade: expressa qual o impacto da ocorrência relatada.
- Prioridade: qual a urgência da atividade solicitada.
- Atribuição: a quem essa ocorrência é direcionada.
- Resumo: titulo ou nome que remeta a ocorrência
- Descrição: deve conter um relato mais aprofundado da ocorrência.
- Visibilidade: se pode ou não ser visto publicamente.

O Mantis trabalha com vários tipos de usuários. O nível de acesso desses usuários é definido de acordo com a função que exercem no projeto, de acordo com a tabela a seguir:

Tabela 2. Usuário e níveis de acesso

NÍVEIS DE ACESSO DOS USUÁRIOS	
VISUALIZADOR	Visualiza casos.
RELATOR	Visualiza, cria e monitora casos.
ATUALIZADOR	Visualiza, cria, monitora e atualiza casos.
DESENVOLVEDOR	Gerencia casos.
GERENTE	Gerencia projetos; Gerencia casos.
ADMINISTRADOR	Gerencia projetos; Gerencia casos; Gerência usuários; Gerencia configurações do sistema.

O ciclo de vida de uma mudança pode sofrer alterações, mas normalmente acontece da seguinte forma:

O relator descreve um novo caso, que é enviado com o status de Novo. O gerente é notificado pelo Mantis e atribui o caso a um desenvolvedor, onde o mesmo é notificado. O desenvolvedor pode solicitar esclarecimentos ao relator e a mudança é

retornada ao status de Retorno. O desenvolvedor analisa e altera, se nenhum retorno for necessário, a mudança é colocada com o status Resolvido, onde o relator e o gerente são notificados. [Matos, R. 2010] A figura 3 a seguir representa exatamente esse ciclo de vida de uma mudança.

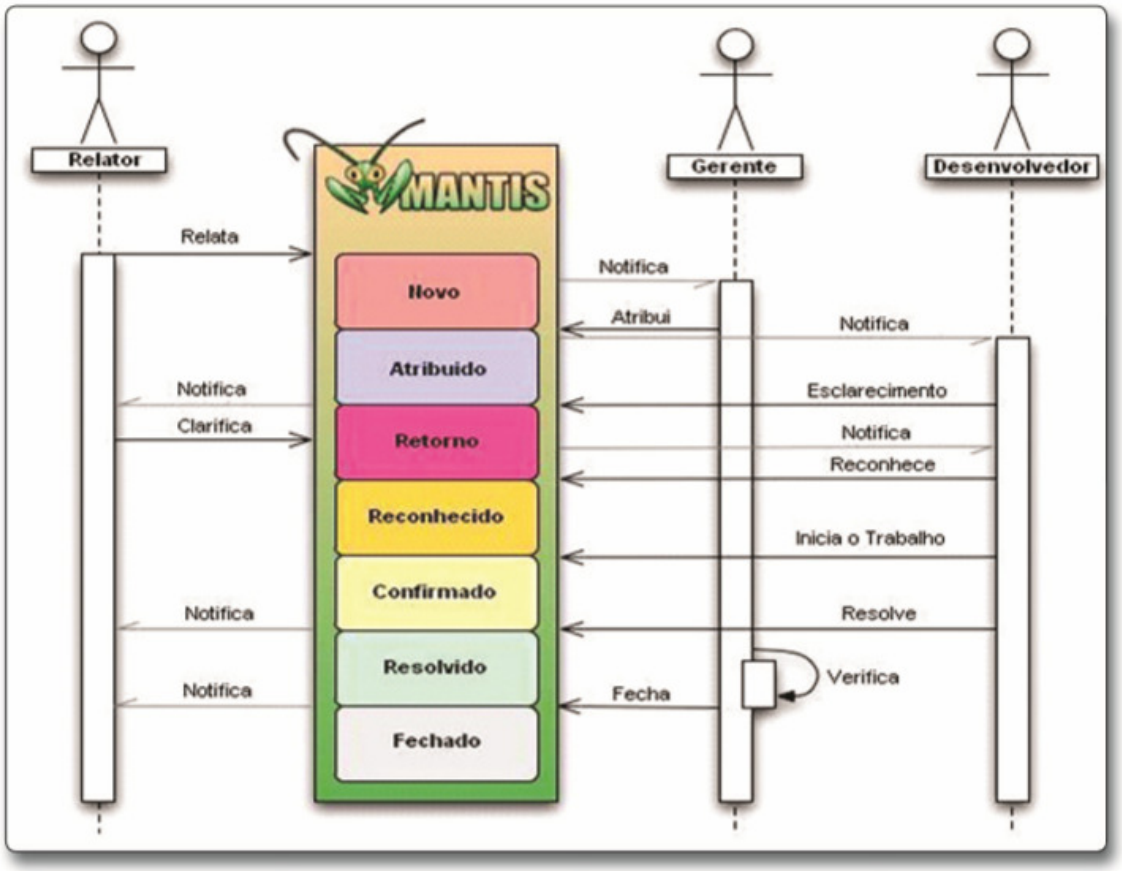

Figura 3. Ciclo de vida de uma mudança

Uma tarefa aberta como "novo" pode ser retornada para quem a abriu, e em seguida pode ser admitida por algum desenvolvedor, o qual pode retornar a tarefa. Após admitida, pode ser atribuída até ser fechada seguindo sequencialmente os estados resolvido, confirmado e fechado.

5.2. Redmine

Uma ferramenta de controle de mudanças extremamente flexível, que permite vários projetos em paralelo. Possui uma *Wiki* embutida, controle de tempo, campos personalizados, integração com os sistemas de controle de versão como o GIT e SVN. Contém calendário e gráficos de *Gantt* para representar os projetos e seus *deadlines* (prazos de entrega), além disso, é facilmente customizável através de *plugins* que podem ser baixados dentro da própria ferramenta [http://www.redmine.org/].

O Redmine, assim como o Mantis, permite que durante os testes os desenvolvedores tomem conhecimento das correções necessárias a ferramenta, o que faz com que as correções possam ser executadas em paralelo ao desenvolvimento do sistema.

Cada projeto pode conter um número N de tarefas. O sistema permite a criação de outros tipos de tarefas, mas comumente elas são classificadas em três tipos: defeito, funcionalidade e suporte, sendo que esta ferramenta especifica permite desabilitar qualquer um dos tipos para determinados projetos que forem desnecessários. Todo projeto tem que possuir nome e identificador, podendo ser acrescidos de algumas características não obrigatórias.

São requisitos obrigatórios para abertura de uma tarefa o seu tipo, título, situação (nova, em progresso, *feedback*, rejeitada, resolvida e encerrada), prioridade e data de inicio, bem como um relator, que é o usuário que a registrou no sistema.

As tarefas podem ou não serem atribuídas a algum usuário. Apesar de estarem divididos de acordo com o papel que cada um desempenha, os usuários podem ter suas permissões completamente alteradas criando permissões personalizadas. Por esta razão torna-se mais difícil criar um fluxo mais complexo do funcionamento do sistema.

5.3. TRAC

Apesar de o TRAC ser usado no gerenciamento de projetos, solicitação de clientes, sua função inicial é controle de mudanças. Assim como outras ferramentas o TRAC também oferece uma *Wiki* integrada além de integração com os sistemas de controle de versão GIT e SVN. A ferramenta também possui uma linha do tempo que facilita o acompanhamento do progresso das atividades que estão sendo desenvolvidas [http://trac.edgewall.org/].

O controle de mudanças é feito através da abertura de tíquetes que são direcionados a um usuário, que após realizar suas anotações pode encerra-lo ou redireciona-lo a outro usuário, gerando assim um histórico de todas as alterações realizadas ao longo do ciclo de vida de um software.

Os tíquetes também são classificados pelo seu tipo. Geralmente usa-se defeito, suporte e funcionalidade, mas a ferramenta permite exclusão ou adição de tipos de tíquetes. Também é permitido ao administrador adicionar ou excluir da criação de tíquetes as propriedades: gravidade, prioridade, versão, fazendo com que as únicas propriedades obrigatórias a abertura de um novo tíquete sejam sumário, relator e dono.

Assim como as propriedades de abertura de tíquetes podem ser alteradas pelo administrador, as permissões de usuário também podem ser personalizadas o que torna a ferramenta muito difícil de modelada.

Bem parecido da forma que o Redmine e o Mantis usam os projetos, a o TRAC, usa marcos para agrupar seus tíquetes. É possível ter mais de um marco aberto paralelamente. Para sua abertura, é preciso nome e prazo final para o encerramento deste marco.

6. Aplicação da Engenharia de Domínio para Descoberta de Padrões de Sistemas de Controle de Mudanças

Esta seção tem por objetivo descrever padrões de análise para as aplicações de controle de mudanças. Foi utilizada a análise de domínio para identificar esses padrões. Os padrões foram separados por assunto, sendo eles Usuário, Projeto e Tarefa.

Espera-se que ao final da análise os resultados obtidos possam contribuir para o desenvolvimento de novas aplicações deste mesmo domínio.

Alguns atributos não foram retratados nos diagramas a seguir por não serem considerados indispensáveis para o bom funcionamento dos sistemas.

6.1. Esquema Perfil de Usuário

A figura 4 apresenta um diagrama de classes que compõem a estrutura necessária para identificar o usuário, bem como seu perfil dentro das ferramentas que foram analisadas: Redmine, TRAC e Mantis.

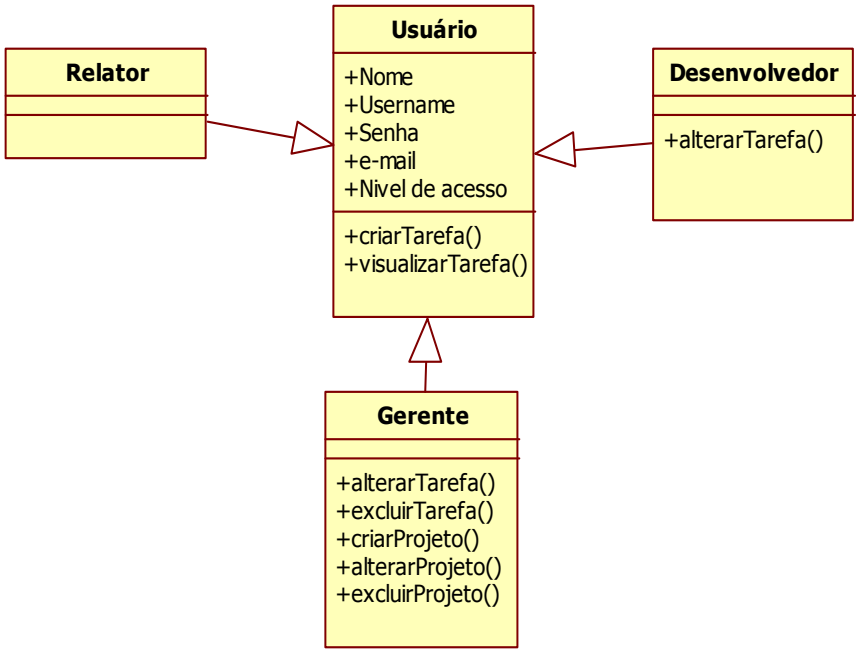

Figura 4. Diagrama de classe que representa o padrão Perfil de Usuário

Pequenas diferenças são encontradas nos dados do usuário entre as ferramentas pesquisadas. Tais características podem ser consideradas opcionais ou adicionais, não sendo assim necessário implementá-las.

Entre todas as ferramentas analisadas foram observados diferentes papeis que um usuário pode assumir. Tanto o Mantis quanto o Redmine apresentaram os seguintes perfis em comum: relator, desenvolvedor e gerente.

Sendo o gerente responsável por administrar os projetos e tarefas tendo total controle sobre eles. O desenvolvedor por sua vez, possui permissão para criar, alterar e visualizar tarefas. Já o relator pode somente criar e visualizar tarefas dentro do escopo dos sistemas analisados.

Apesar das classes Relator, Desenvolvedor e Gerente possuírem os mesmos atributos herdados da classe usuário, elas diferenciam-se entre si através de seus métodos, que são também o que caracterizam as permissões entre os diferentes perfis de usuário já descritos acima.

6.2. Esquema Projeto

A figura 5 representa a estrutura usada pelo Trac, Redmine e Mantis onde um gerente cria, altera e exclui projetos.

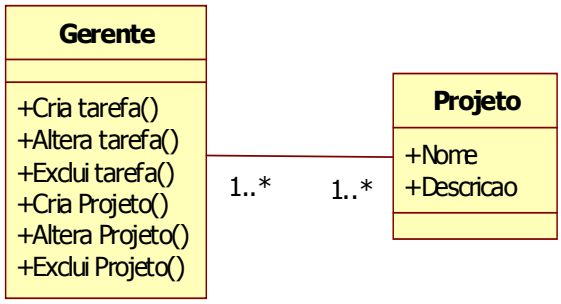

Figura 5. Diagrama de classe que representa o padrão Projeto

Nas ferramentas analisadas um projeto não pode ser criado, alterado ou excluído pelos perfis relator e desenvolvedor. Somente o perfil de usuário gerente tem permissão para manipular os dados desta classe.

É importante ressaltar que um perfil de usuário gerente pode abrir vários projetos, para que os outros perfis interajam no mesmo, adicionando tarefas para o fluxo do sistema. Apesar de somente um gerente conseguir cadastrar um projeto, nada impede que outros usuários com esta mesma permissão possam manipula-lo.

O que a classe projeto tem em comum além de sua relação direta com a classe usuário são os atributos nome e descrição. Fica claro nas ferramentas utilizadas que para existir um projeto, não se faz necessário que exista uma tarefa associada, o que não acontece na situação contraria.

6.3. Esquema Tarefa

Existem algumas variações de campos para criação de tarefas. Essas características, porem, podem ser consideradas opcionais por não se fazerem obrigatórias para o cadastro de uma nova tarefa.

No entanto, tanto o TRAC, o Redmine e o Mantis apresentam como requisitos comuns para abertura de uma nova tarefa os atributos: projeto, nome, descrição, tipo, prioridade, atribuição, arquivo.

Apesar dos atributos arquivo e atribuição, presente nas três ferramentas analisadas, não serem obrigatórios, eles fazem parte da estrutura obrigatória de uma tarefa, representando informações consideradas indispensáveis, mesmo assumindo valores zerados.

A figura 6 apresenta uma estrutura que relaciona projeto e tarefa, onde um projeto pode possuir várias tarefas e uma tarefa pode pertencer a somente um projeto.

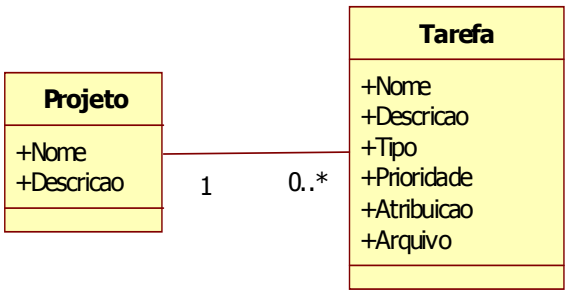

Figura 6. Diagrama de classe que representa o padrão Tarefa

7. Considerações Finais

Através de técnicas de reuso, análise de domínio e análise de padrões, este trabalho vem tentar descrever esquemas conceituais das ferramentas de controle de mudanças com interface web, a fim de facilitar o entendimento da estrutura para criação de novas ferramentas deste segmento. Para representação dos padrões foi usado UML que além de ser uma linguagem mais recente não limita a identificação desses esquemas e posterior padrões a método algum.

Três ferramentas web foram analisadas durante toda a construção deste trabalho. Tanto o Mantis, o Redmine e o TRAC apresentam uma grande possibilidade de personalização da ferramenta, o que dificultou um pouco mais a descrição desses padrões, mas por outro lado também permite ao usuário gerente do sistema, manipular e adequar a ferramenta para que possa trabalhar somente com as informações que julga de seu interesse.

Levando em consideração o grande número de atributos possíveis as classes relatadas acima, tentou-se observar a forma mais genérica possível, mas ainda assim mantendo toda funcionalidade que o sistema propõe.

Além das características citadas, o sistema também possui outras características não abordadas neste trabalho, mas que futuramente poderiam ser propostas a fim de que venham complementar esse modelo e facilitar ainda mais a construção de novos sistemas do mesmo domínio. A formalização e catalogação desses padrões também são sugestões de futuros trabalhos.

8. Referências

Almeida, E.S., Alvaro, A., Garcia, V. C., Mascena, J. C. C. P., Burégio, V. A. A., Nascimento, L. M.; Lucrédio, D, Meira, S. R. L. (2007) C.R.U.I.S.E: Component Reuse in Software Engineering. C.E.S.A.R e-book, Brasil.

Arango, G. (1994), Domain Analysis Methods. Software Reusability. Chichester, Ellis Horwood, England.

Atkinson, C., J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B.

Peach, J. Wust, e J. Zettel. (2001). Component-Based Product Line Engineering with UML. Addison-Wesley.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S. (1997) A Pattern Language. Oxford University Press, NY.

Cooper, J. W. (1998) Using Design Patters. CACM, vol. 41.

Coplien, J. O. (1996) Software Patters. SIGS Books, NY.

Cornwell, P.C. (1996) HP domain analysis: Producing useful models for reusable software. Hewlett-Packard. Journal, article 5 de agosto.

Clements, P., e L. M. Northrop. (2001) Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA.

Cunha, G. E. (1999) Proposta de Reuse Patterns para a Reutilização Sistemática em Sistemas de Software Orientado a Objetos. Trabalho de Conclusão de Curso. Centro de Ciências Exatas e Tecnológicas/Universidade do Vale do Rio dos Sinos-UNISINOS.

Dart, S. (1991) Concepts in Configuration Management Systems. In International Workshop on Software Configuration Management (SCM), 1-18. Trondheim, Norway: ACM Press.

Devedzic, V. (1999) Ontologies: Borrowing from Software Patterns. Intelligence, Fall.

Fowler M. (1997) Analysis Patterns: Reusable Object Patterns, Addison-Wesley.

Fowler M., Scott, K. (2000) UML Essencial, Um breve guia para a linguagem padrão de modelagem de objetos. 2° edição Bookman, Porto Alegre.

Frakes, William B., e Kyo Kang. (2005) Software Reuse Research: Status and Future. IEEE Transactions of Software Engineering 31, n. 7: 529-536.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.

Mantis. (2015) Disponível em: http://www.mantisbt.org/. Acesso em: 08 de maio.

Matos, R. (2010) Gestão da Manutenção e Configuração de Software através do controle de mudanças. Edição 20 - Engenharia de Software Magazine.

Monteiro, A. J. B. (2002) Padrões de Análise em Aplicações de Comércio Eletrônico na Web. Dissertação de Mestrado em Ciência da Computação, Universidade Federal de Minas Gerais.

Neighbors, J. (1989) Draco: A Method for Engineering Reusable Software Systems. In Software Reusability: Concepts and Models, 1:295-319. ACM Press Frontier Series. Addison-Wesley.

Prieto-Diaz & Arango, (1991) Domain analysis: an introduction. Software Engineering Notes, 15(2):47–54.

Robertson, S., Strunch, K. (1993) Reusing the Products of Analysis. Second International Workshop on Software Reusability.

Redmine (2015) Disponível em: http://www.redmine.org/ Acesso em: 01 de maio.

Salviano, C. F. (1997) Introdução a Software Patterns. Fundação Centro Tecnológico para Informática - CTI. Campinas, São Paulo.

Silva, E. O. (2008) Integração De Padrões De Análise E Ontologias De Domínio: Um Estudo De Caso No Domínio De Gestão Urbana. Dissertação Universidade Federal de Viçosa.

Silva, E. O., Lisboa Filho, Jugurta, Oliveira, Alcione de P. Revisando Padrões de Análise para Bancos de Dados Geográficos usando Meta-Propriedades Ontológicas. Departamento de Informática, Universidade Federal de Viçosa.

TRAC (2015) Disponível em: http://trac.edgewall.org/ Acesso em: 21 de maio.

Silva, M., Moreno, A., (2011) Automação em Testes Ágeis. Revista de Sistemas e Computação, Salvador, v. 1, n. 2, p. 139-164, disponível em: http://www.revistas.unifacs.br/index.php/rsc