UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUÍS FERNANDO HECKLER

Redmine GCO: uma ferramenta de apoio para a Gerência de Configuração

Trabalho de Conclusão de Curso

Prof. Dr. Marcelo Soares Pimenta
Orientador

Porto Alegre, dezembro de 2009

"Mulheres e homens, somos os únicos seres que, social e historicamente, nos tornamos capazes de apreender. Por isso, somos os únicos em quem aprender é uma aventura criadora, algo, por isso mesmo, muito mais rico do que meramente repetir a lição dada. Aprender para nós é construir, reconstruir, constatar para mudar, o que não se faz sem abertura ao risco e à aventura do espírito." — PAULO FREIRE, PEDAGOGIA DA AUTONOMIA, p.69

AGRADECIMENTOS

Ao professor Marcelo Soares Pimenta, pela orientação, apoio e confiança.

Aos colegas e amigos Diego Francisco de Gastal Morales e Lia Trodo, pelas conversas diversas e incentivo.

Ao meu irmão Cláudio André Heckler e ao amigo Ricardo Luís Lichtler, pelo exemplo na profissão, incentivo e apoio constantes durante todo este período na universidade.

À empresa iProcess Soluções em Tecnologia Ltda, especialmente na figura de seus colaboradores, pelo apoio, motivação e oportunidade.

Aos meus amigos, pela compreensão pelos momentos em que estive ausente.

À minha segunda família, Thomé da Cruz, especialmente Fátima e Osmar, pela torcida, carinho e, claro, também pelas comidas deliciosas.

Aos meus familiares, meu irmão, minhas irmãs e especialmente meus pais Cláudio e Telmi Heckler, pelo amor, estímulo e amizade constantes.

Por fim, mas não menos importante, à minha amada esposa Fabiana Thomé da Cruz, pelo apoio, companheirismo e amor incondicionais.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Asynchronous Javascript And XML*
API	*Application Programming Interface*
CMMI	*Capability Maturity Model Integration*
GCO	Gerência de Configuração
GCS	Gerência de Configuração de *Software*
GPL	*General Public License*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
MPS.BR	Melhoria do Processo de *Software* Brasileiro
MVC	*Model-View-Controller*
NC	Não-conformidade
SCM	*Software Configuration Management*
SEI	*Software Engineering Institute*
SOFTEX	Sociedade Brasileira para Promoção da Exportação de *Software*
SVN	*Subversion*
TQM	*Total Quality Management*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

As demandas atuais do mercado de desenvolvimento de *software* pressionam as empresas a aumentar a qualidade de seus produtos, ao mesmo tempo que o grau de complexidade dos sistemas é cada vez maior. Como alternativa tem sido utilizados modelos de qualidade para o processo de desenvolvimento como o CMMI e, no Brasil, o MPS.BR. Dentre as práticas presentes nestes modelos estão as atividades de Gerência de Configuração, que requerem ferramentas de apoio para sua efetiva realização. Porém, constata-se uma carência de sistemas que integrem estas e outras funcionalidades e formem um repositório central de informações, especialmente útil para a coleta e registro das evidências necessárias à avaliação formal em tais modelos de qualidade. Este trabalho tem por objetivo implementar uma ferramenta para apoiar as atividades de Gerência de Configuração, integrada com o sistema de gerência de projetos *Redmine*, no âmbito da empresa de desenvolvimento iProcess Soluções em Tecnologia Ltda. Para tanto, apresenta-se os aspectos de planejamento e projeto, bem como o detalhamento da evolução das principais versões desenvolvidas até alcançar-se a solução proposta: a ferramenta *Redmine GCO*. Acredita-se que esta será uma ferramenta que auxiliará especialmente pequenas e médias empresas de desenvolvimento de *software*, aumentando a qualidade de seus processos e, portanto, de seus produtos. Ao final, discute-se as perspectivas futuras e oportunidades de continuidade deste trabalho.

Palavras-chave: Gerência de Configuração, MPS.BR, qualidade de *software*.

Redmine GCO: a tool for Configuration Management

ABSTRACT

The current demands of the software development market enforce companies to increase the quality of their products, at the same time that the system's complexity is increasing. Alternatively it has been used quality models for the development process as CMMI, and in Brazil, MPS.BR. Among the practices present in these models are the activities of Configuration Management, which require support tools to take effect. However, there is a lack of systems that integrate these and other features and form a central repository of information, especially useful for collecting and control necessary evidences for the formal appraisal in such quality models. This work aims to develop a tool to support configuration management activities, integrated with the project management system Redmine, within the development company *iProcess Soluções em Tecnologia Ltda.* It presents the planning and design aspects, as well as the detailed evolution of major versions developed to reach the solution proposed: the Redmine GCO tool. It is believed that this will be a tool that will help especially small and medium-sized software development companies, increasing the quality of its processes and thus their products. Finally, it is discussed future prospects and opportunities for continuing this work.

Keywords: Configuration Management, MPS.BR, software quality.

1 INTRODUÇÃO

Na atual conjuntura do mercado de desenvolvimento de *software*, a qualidade deixa de ser um mero fator coadjuvante para passar a ser um diferencial essencial. Neste contexto, as empresas que desejam manter-se competitivas frente às instabilidades do mercado devem buscar meios de garantir que a qualidade perpasse todas as etapas da produção de seus sistemas, desde a prospecção de novos clientes, levantamento e análise de requisitos até a entrega do software e posterior manutenção.

Neste sentido, as práticas e modelos de qualidade tais como o CMMI (no original *Capability Maturity Model Integration*), proposto pela instituição norte-americana *Software Engineering Institute* (SEI), e, especificamente no caso deste trabalho, o MPS.BR (Melhoria do Processo de *Software* Brasileiro), proposto pela Sociedade Brasileira para Promoção da Exportação de *Software* (SOFTEX), podem ser vistos como ferramentas de apoio à sistematização das atividades do processo de desenvolvimento de *software* com vistas à garantia da qualidade. Porém, a aplicação de tais modelos exige a adoção de uma série de práticas e, se a empresa desejar submeter-se a uma avaliação formal para verificar seu nível de aderência a tais modelos, deverá apresentar elementos que atestem o seu nível de maturidade. Estas provas são denominadas "evidências" pela comunidade da área de qualidade, e sua coleta pode ser muito mais fácil e eficiente caso a empresa disponha de ferramentas apropriadas, que auxiliem na centralização das informações sem todavia acarretar um aumento de burocracia em seus processos.

No caso de pequenas e médias empresas estima-se que os investimentos em implantar tais práticas seja sensivelmente mais representativo em seu orçamento e planejamento estratégico, sendo, portanto, necessário buscar alternativas que tornem tais investimentos viáveis. Assim, um dos pontos a observar é a possibilidade de utilizar ferramentas que auxiliem a gerência de configuração de todos os artefatos produzidos como resultado das diversas etapas de desenvolvimento (não apenas código-fonte mas também documentos de especificação, diagramas, atas de reuniões, etc.) de forma a permitir a coleta de evidências do cumprimento das obrigações requisitadas pelo modelo de qualidade adotado pela empresa. Ao mesmo tempo, constata-se que há carência de ferramentas de licenciamento livre (e portanto sem custo de aquisição) que atendam de modo efetivo e completo tais demandas, principalmente no âmbito da implantação do modelo MPS.BR na empresa de desenvolvimento iProcess Soluções em Tecnologia Ltda[1], alvo de estudo neste trabalho.

[1]Ver http://www.iprocess.com.br

1.1 Motivação

Este trabalho tem como motivação principal prover uma solução que atenda as demandas de Gerência de Configuração no contexto supracitado, sem custo de aquisição e portanto de livre acesso. Espera-se desta forma contribuir para que mais empresas de desenvolvimento de *software*, especialmente de pequeno e médio porte, tenham acesso à ferramentas que as auxiliem a atingir padrões de qualidade cada vez maiores em seus sistemas e produtos, segundo as melhores práticas do mercado e academia.

1.2 Objetivo

Tem-se por objetivo geral estender a ferramenta de gerência de projetos *Redmine*[2], adicionando funcionalidades que favoreçam a automação de atividades de Gerência de Configuração demandadas pela adoção do modelo de qualidade MPS.BR[3]. Desta forma, consolida-se esta ferramenta como um ambiente integrado para o controle da qualidade dos projetos e repositório de "evidências" das práticas relacionadas.

Ainda, como objetivo secundário, espera-se o aprimoramento e consolidação da experiência do presente estudante no processo de elicitação de requisitos, busca e análise de ferramentas, projeto e implementação da solução.

1.3 Estrutura do trabalho

O restante deste relatório estrutura-se conforme descrito a seguir.

O capítulo 2 aborda as definições conceituais acerca do tema de qualidade de *software*, iniciando com uma visão mais generalizada para então restringir-se o foco, passando pela localização do tema de Gerência de Configuração junto à área de Engenharia de *Software*, sua relação na contribuição para a melhoria da qualidade, até alcançar-se uma breve discussão sobre modelos de melhoria de processos de desenvolvimento. São apresentados os conceitos gerais dos modelos CMMI e MPS.BR.

O capítulo 3 inicia-se com a apresentação do contexto que levou ao desenvolvimento deste trabalho, o levantamento dos requisitos e a definição do escopo a ser desenvolvido. A seguir relata-se a pesquisa realizada na busca por soluções que atendessem a demanda observada. A partir do resultado desta pesquisa optou-se pelo desenvolvimento de uma solução específica, cujos pormenores de projeto, arquitetura e planejamento são descritos no restante do capítulo.

A descrição e apresentação dos módulos e das funcionalidades que compõem a solução desenvolvida é feita no capítulo 4, detalhando-se a evolução desde as principais versões intermediárias até o produto final.

Por fim, o capítulo 5 relata as considerações finais sobre o trabalho desenvolvido, sugestões de melhorias identificadas e perspectivas futuras.

Detalhes técnicos dos pré-requisitos dos componentes utilizados e instruções de instalação estão disponíveis no apêndice A ao final deste relatório.

---

[2] Ver http://www.redmine.org/

[3] A escolha de tal ferramenta deve-se ao fato da mesma ser atualmente a principal ferramenta de apoio ao processo de desenvolvimento em uso na empresa alvo deste estudo, iProcess Soluções em Tecnologia Ltda.

2 QUALIDADE DE SOFTWARE E GERÊNCIA DE CONFIGURAÇÃO: REVISÃO BIBLIOGRÁFICA

Para compreender a motivação apresentada na seção 1.1 e o contexto tecnológico envolvido no tema deste trabalho, primeiramente é preciso consolidar os principais conceitos que tangem o tema de Gerência de Configuração (GCO), incluindo a própria definição deste. Para tanto, neste capítulo apresenta-se primeiramente as dimensões e conceitos da qualidade, revisita-se a área de Engenharia de *Software* e dentro desta a disciplina de Gerência de Configuração. Por fim, introduz-se o tema de melhoria de processos de desenvolvimento e apresenta-se os modelos CMMI e MPS.BR, com foco nas questões de GCO.

2.1 Dimensões e conceitos da qualidade

Quando nos referimos ao termo *qualidade* é preciso contextualizar o enfoque pretendido, uma vez que abre-se um leque de definições muito amplo a cerca deste que é um conceito subjetivo, mas presente no dia-a-dia das pessoas (KAN, 2002). Por exemplo, pode-se pensar em qualidade de vida da população, qualidade da água e ar do ambiente, qualidade da comida de um restaurante ou ainda, de forma mais genérica, qualidade de produtos e serviços dos quais se é consumidor ou fornecedor.

Tem-se uma definição popular no dicionário Houaiss da Língua Portuguesa que define qualidade como a "capacidade de atingir os efeitos desejados; propriedade" (HOUAISS; VILLAR, 2004). Buscando uma visão técnica, o Glossário Padrão de termos de Engenharia de *Software* define qualidade como "o grau com que um sistema, componente ou processo satisfaz seus requisitos e expectativas ou necessidades de seus consumidores e usuários" (IEEE, 1990, p.60). Complementando esta definição, MAGALHãES (2008, p.1) considera que a definição de qualidade "engloba tanto a qualidade do produto (conformidade com os requisitos) quanto a qualidade do processo (grau em que o processo garante a qualidade do produto). Sua definição e aplicação, porém, se modifica em função do domínio no qual é tratada."

No caso de produtos de *software*, onde lida-se com um ente artificial, o desafio de mensurar (e comparar) suas características é sem dúvida muito maior que no caso de objetos do mundo físico, mas ainda assim é factível, como coloca PRESSMAN (2001). Segundo o autor, dentre as propriedades que pode-se observar em um *software* pode-se citar a complexidade ciclomática, coesão, número de pontos de função, número de linhas de código, etc. (PRESSMAN, 2001, p.195).

Então, de um modo geral pode-se estabelecer que, se um dado objeto possui características mensuráveis, um dos modos de medir a qualidade deste objeto é

comparando a avaliação destas características em relação ao projeto do mesmo. É o que PRESSMAN (2001, p.195) define como "qualidade de *design*". Neste sentido CROSBY (1979) *apud* GUERRA; COLOMBO (2009, p.45) afirma: "Qualidade é conformidade com os requisitos" e portanto estes devem estar definidos de forma a poderem ser gerenciados por meio de medidas apropriadas, reduzindo-se o retrabalho e aumentando a produtividade. As autoras enfatizam ainda que é preciso focar a melhoria da qualidade nos processos da organização, para além das pessoas. A qualidade é responsabilidade de todos (GUERRA; COLOMBO, 2009). Este foco no processo de implementação é o que PRESSMAN (2001, p.195) coloca como "qualidade de conformidade".

Tem-se então que a qualidade do produto de *software* implica diretamente no custo de produção do mesmo, podendo levar a prejuízos consideráveis às organizações (BARTIE, 2002). Em seu guia geral do modelo de Melhoria do Processo do *Software* Brasileiro a SOFTEX é enfática em definir que "assim como para outros setores, qualidade é fator crítico de sucesso para a indústria de *software*" (SOFTEX, 2009a, p.6).

Uma vez sendo a qualidade tão importante, sua busca e estudo deixam o empirismo a partir do momento em que se torna um dos objetivos de estudo da Engenharia de *Software*, conforme descrito na próxima seção.

2.2 Engenharia de *Software*

Sistemas de computador (*softwares*) são desenvolvidos almejando-se que se tornem úteis de forma a transformar para melhor a situação de trabalho das pessoas, seus usuários. Além disso, espera-se que sejam fáceis de usar, estáveis ao longo do tempo e facilmente alteráveis quando for necessário modificá-los. Quando alguma destas características não é alcançada pode-se dizer que o sistema não foi bem sucedido. Desta forma, certamente o intuito de todo desenvolvedor de sistemas é que seus *softwares* sejam bem sucedidos, ajudem as pessoas, tornando melhor o ambiente em que são aplicados. E para alcançar tal objetivo é preciso ter rigor, método e disciplina, tanto na etapa de projeto como no decorrer do desenvolvimento do sistema (PRESSMAN, 2001). Segundo o autor, o trabalho de construção de um *software* é composto basicamente por três fases: definição, desenvolvimento e suporte. No desempenho destas fases tipicamente são desenvolvidas atividades de diferentes naturezas:

- Controle e rastreabilidade do projeto
- Revisões técnicas formais
- Garantia da qualidade
- Gerência de configuração
- Preparação e produção de documentação
- Gerência de reuso
- Medição
- Gerência de riscos

Cada uma destas atividades, além de outras não citadas pelo autor, são de alguma forma objeto de estudo da Engenharia de *Software*, que busca desenvolver e propor técnicas e metodologias que auxiliem os desenvolvedores (sejam grandes empresas ou programadores individuais) a construírem sistemas de forma melhor, mais fácil, mais eficiente e com mais qualidade. Dentre as atividades citadas, para o contexto deste trabalho, serão focadas aquelas relacionadas à gerência de configuração.

2.2.1 Gerência de Configuração

O desenvolvimento de um *software* implica diretamente na criação de diversos artefatos, que são os produtos resultantes das diversas atividades realizadas ao longo do desenvolvimento: programas executáveis, códigos-fonte, modelos, diagramas e especificações, bem como outros. Boa parte destes artefatos não apenas são criados no decorrer do projeto como são alterados, evoluem ao longo do tempo de desenvolvimento. Organizar e controlar o ciclo de vida de todos os artefatos do projeto é o objetivo da área de "Gerência de Configuração de *Software*"(GCO, GCS em algumas referências ou, no original em inglês: *Software Configuration Management - SCM*)[1] (PAULA FILHO, 2003).

Para implementar a *configuração* de um projeto, por menor que seja, é preciso no mínimo possuir controle de versão dos artefatos. Por meio desse controle é possível ter acesso às versões antigas que contém material que pode ser reaproveitado, permite-se estabelecer pontos estáveis do sistema e voltar facilmente para estes pontos para, por exemplo, poder desfazer uma ação que tenha trazido inadvertidamente como versão mais atual uma versão antiga ou defeituosa (PAULA FILHO, 2003).

Porém é importante ressaltar que a GCO vai muito além do controle de versões. Suas atividades relacionadas incluem a identificação e estabelecimento de linhas-base (*baselines*); revisão, aprovação e controle de mudanças; rastreabilidade e divulgação de mudanças; auditorias e revisões do produto de *software*; gerenciamento de versões (*releases*) (IEEE, 2005, p.5).

Tem-se portanto que as atividades de GCO constituem ferramenta fundamental no desenvolvimento de *software* de qualidade. Na seção 2.3 a seguir descreve-se alguns modelos de melhoria de processo de desenvolvimento atualmente em uso no Brasil e que têm entre suas atividades questões diretamente endereçadas à GCO.

Antes de prosseguir porém, de forma a compreendermos melhor os capítulos seguintes deste trabalho, é importante formalizarmos alguns conceitos relacionados ao tema de GCO, conforme segue:

Itens de Configuração

Produtos de trabalho do processo, também chamados de artefatos, tais como documentos, planos de teste, código-fonte, etc., controlados durante o ciclo de vida do sistema (IEEE, 2005).

Baseline

Trata-se do conjunto de itens de configuração formalmente designados e fixados em um dado tempo do ciclo de vida do *software*. Durante o processo de desenvolvimento são estabelecidas *baselines* (tipicamente mais de uma) de forma a garantir

[1] No restante deste trabalho, quando for citada a sigla GCO, estará sendo feita referência à "Gerência de Configuração" ou "Gerência de Configuração de *Software*" de forma equivalente.

um fluxo ordenado do trabalho (BUCKLEY, 1996).

Plano de Configuração

O Plano de Gerência de Configuração, ou simplesmente Plano de Configuração, é o documento onde identifica-se os itens de configuração e o planejamento do controle destes em *baselines*, bem como os critérios utilizados para a seleção dos mesmos (SOFTEX, 2009b).

Controle de versões

Denomina-se por "controle de versões" o conjunto de procedimentos e ferramentas utilizados para gerenciar as diferentes versões dos itens de configuração ao longo do ciclo de vida do sistema (PRESSMAN, 2001). Sua implementação está associada à criação de um "repositório" onde são armazenados os artefatos, permitindo a promoção de novas versões e resgate de versões anteriores[2].

Revisão

Dentro do sistema de controle de versões, revisão é a denominação de cada versão do repositório. Desta forma, uma revisão pode estar associada a um ou mais artefatos sob controle.

Controle de Mudanças

Diretamente relacionado ao controle de versões, o controle de mudanças estabelece os critérios e mecanismos para a promoção ou revogação de uma dada versão do *software*, controle de acesso e controle de sincronização dos itens de configuração (PRESSMAN, 2001).

Auditoria de configuração

Trata-se do processo de verificação da conformidade e consistência do estado atual dos itens de configuração em relação ao planejado (PRESSMAN, 2001).

2.3 Modelos de melhoria do processo de desenvolvimento

Tomando novamente o glossário de termos da IEEE entende-se que *processo* caracteriza-se como uma "seqüência de passos realizados para atingir um determinado propósito" (IEEE, 1990, p.57). No contexto do desenvolvimento de *software* tem-se então que o "processo de desenvolvimento" nada mais é que a seqüência de passos tomados tendo como objetivo a implementação, disponibilização e manutenção de um dado sistema. Porém é preciso compreender que a simples existência de um processo não garante a qualidade deste e tampouco do produto final pretendido. Como coloca BROOKS (2004), não existe uma *bala de prata*, uma solução, ferramenta ou método que assegure a produtividade e qualidade a baixos custos no desenvolvimento de *software*.[3] Por outro lado é crescente a pressão de mercado para

[2]Uma visão geral sobre sistemas de controle de versões, suas características, componentes, modos de uso e principais ferramentas disponíveis pode ser vista em http://pt.wikipedia.org/wiki/Sistema_de_controle_de_versão.

[3]A edição citada é uma edição de aniversário, mais recente, onde o autor revisita alguns dos conceitos e afirmações da versão original. Mas, de fato, percebe-se que de um modo geral o dilema

que as empresas atendam às demandas de forma mais ágil, com menor custo e com mais qualidade, ao mesmo tempo que os sistemas alcançam um grau de complexidade cada vez maior (CHRISSIS; KONRAD; SHRUM, 2003).

Frente a tais desafios, por meio dos estudos da Engenharia de *Software*, têm surgido abordagens de organização do processo de desenvolvimento de forma a buscar atender estas demandas. Procura-se trazer garantias de que cada etapa do processo gere um produto intermediário que satisfaça os requisitos da etapa seguinte, de forma que ao final do processo o produto final atenda os requisitos especificados. Para conseguir aumentar a qualidade desta maneira, não basta que se tenha um processo de desenvolvimento qualquer, mas sim que modelos e técnicas sejam seguidos, fazendo-se uso de ferramentas e tecnologias adequadas (KAN, 2002).

Os primeiros esforços neste sentido surgiram a partir da adequação e uso da filosofia de Qualidade Total (TQM no original). A partir de então surgiram *frameworks* organizacionais buscando o aumento da qualidade, tais como *PDCA - Plan-Do-Check-Act*, *Quality Improvement Paradigm/Experience Factory Organization* e *Software Engineering Institute (SEI) Capability Maturity Model*. Percebe-se que os modelos de processo de desenvolvimento têm evoluído para abordagens cada vez mais iterativas e proativas no sentido de buscar evitar ou diminuir a incidência de defeitos. Partiu-se do modelo cascata, prototipação, espiral, iterativo, processo unificado, processo de prevenção de defeitos, até o estabelecimento dos modelos de maturidade atuais (KAN, 2002).

Dos modelos recentes, basicamente dois se destacam no cenário brasileiro: o CMMI - *Capability Maturity Model Integration* desenvolvido pelo SEI; e o MPS.BR - *Melhoria de Processo de Software Brasileiro* proposto pela SOFTEX, que serão abordados a seguir[4].

2.3.1 CMMI

O CMMI foi criado a partir da junção de três modelos de maturidade criados anteriormente - *Capability Maturity Model for Software - SW-CMM*; *Systems Engineering Capability Model - SECM*; e *Integrated Product Development Capability Maturity Model - IPD-CMM*. Porém esta integração não foi apenas uma simples união de cada uma das recomendações mas sim uma evolução de forma a compor um *framework* capaz de trazer benefícios a todas as etapas do processo de desenvolvimento nas organizações (CHRISSIS; KONRAD; SHRUM, 2003). Sua estrutura é organizada em quatro disciplinas ou áreas de conhecimento:

- Engenharia de sistemas;
- Engenharia de *software*;
- Desenvolvimento integrado de produto e processo;
- Contratação de fornecedores (no original, *supplier sourcing*).

Cada disciplina possui uma série de "áreas de processo", que constituem-se de boas práticas que, quando implementadas em conjunto, satisfazem uma série de objetivos específicos e considerados importantes para que de fato tais áreas sejam

permanece o mesmo: desenvolver *software* é complexo, o que deixa margem para que possam ocorrer muitos problemas e dificuldades.

[4]Não é escopo deste trabalho realizar uma revisão profunda dos modelos.

melhoradas significativamente. Cada área de processo possui componentes obrigatórios (devem ser implementados pelo processo), componentes esperados (tipicamente implementados, usados como guia para alcançar a implementação dos itens obrigatórios) e os componentes informativos (ajudam a entender e aplicar os demais componentes) (CHRISSIS; KONRAD; SHRUM, 2003).

Não é esperado que uma organização implemente um processo de desenvolvimento que contemple a cobertura completa de todas as áreas de processo das disciplinas selecionadas em uma única etapa de implantação do modelo. Como o próprio nome especifica, a organização e seu processo devem adquirir maturidade, o que é conquistado aos poucos. Desta forma, o modelo estabelece níveis de maturidade e capacidade, dependendo de como está sendo implementado. Quando aplica-se o modelo em sua representação contínua considera-se os níveis de capacidade, que são aplicados na melhoria dos processos da organização em cada área de processo. Já quando aplica-se o modelo em sua representação estagiada, considera-se os níveis de maturidade, que por sua vez são aplicados na melhoria dos processos através de múltiplas áreas de processo (SAMARANI, 2005). A Tabela 2.1 a seguir demonstra comparativamente os níveis de capacidade e maturidade do modelo.

Tabela 2.1: Níveis do CMMI

Nível	**Representação contínua**	**Representação por estágios**
	Níveis de Capacidade	**Níveis de Maturidade**
0	Incompleto	-
1	Executado	Inicial
2	Gerenciado	Gerenciado
3	Definido	Definido
4	Quantitativamente gerenciado	Quantitativamente gerenciado
5	Em otimização	Em otimização

Fonte: SAMARANI (2005, p.15)

Para alcançar o nível 2 do modelo (CMMI-N2) o processo de desenvolvimento deve atender, entre outros, os objetivos da área de processo "Gerência de Configuração", conforme segue:

1. Estabelecer *baselines*
   - Identificar itens de configuração
   - Estabelecer um sistema de gerência de configuração
   - Criar ou liberar as *baselines*
2. Rastrear e controlar mudanças
   - Rastrear as requisições de mudanças
   - Controlar os itens de configuração
3. Estabelecer integridade
   - Estabelecer registros de gerência de configuração
   - Executar auditorias de configuração

Como o modelo estabelece que para alcançar-se um nível superior é preciso continuar atendendo plenamente os requisitos dos níveis anteriores, tem-se então que as práticas de GCO devem ser aplicadas em todos os níveis do modelo.

2.3.2 MPS.BR

O programa MPS.BR foi criado com o objetivo de ser adequado especialmente para pequenas e médias empresas de desenvolvimento, públicas ou privadas (embora seja adequado também para grandes empresas). Além disso, foi criado buscando compatibilidade com os padrões de qualidade internacionais já estabelecidos e aceitos, aproveitando as competências dos modelos e padrões existentes. Da mesma forma como o CMMI, o MPS.BR baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade dos *softwares* e serviços relacionados desenvolvidos. Trata-se de um modelo genuinamente brasileiro, cuja fundamentação técnica está baseada nas normas NBR ISO/IEC 12207 e ISO/IEC 15504, e em conformidade com o modelo CMMI-DEV (versão atual do modelo CMMI) (SOFTEX, 2009a).

Assim como o CMMI, o MPS.BR possui uma divisão em estágios, chamados níveis de maturidade. Esta divisão, embora baseada na divisão de níveis do CMMI, possui um maior número de estágios com o objetivo de tornar a transição entre os níveis mais adequada ao porte de pequenas, médias e até mesmo micro empresas, diminuindo os custos e proporcionando uma visibilidade dos resultados em prazos mais curtos (SOFTEX, 2009a). A Tabela 2.2 a seguir demonstra a relação entre os níveis de ambos os modelos.

Tabela 2.2: Relação entre os níveis do CMMI e MPS.BR

Níveis MPS.BR	**Níveis CMMI**
G	2
F	
E	3
D	
C	
B	4
A	5

Fonte: SOFTEX (2009)

O MPS.BR é dividido em processos e estes são descritos em termos de seu propósito e resultados esperados. A capacidade de cada processo é representada segundo diversos atributos de processo, expressando o grau de refinamento e institucionalização com que o processo é executado na organização. Desta forma, a cada nível de maturidade são definidos processos que devem ser implementados em um dado nível de capacidade, segundo os atributos que devem ser atendidos e os resultados esperados que devem ser obtidos. Conforme verifica-se na tabela 2.3 a seguir, que apresenta os níveis de maturidade e os processos correspondentes a cada nível, o MPS.BR coloca as atividades de GCO como obrigatórias a partir do nível F. Na prática, no entanto, a organização começa a implementar atividades de GCO já no nível G por conta da necessidade de manter a rastreabilidade bidirecional entre os requisitos e os produtos de trabalho. Ainda, assim como no CMMI, para implementar todos os requisitos de um dado nível do modelo é preciso continuar implementando

todos os requisitos dos níveis abaixo. Desta forma, as atividades de GCO passam a ser obrigatórias a partir do nível F até o nível A.

Tabela 2.3: Níveis de maturidade e processos do MPS.BR

Nível	**Processos**
A	
B	Gerência de Projetos – GPR (evolução)
C	Gerência de Riscos – GRI
	Desenvolvimento para Reutilização – DRU
	Gerência de Decisões - GDE
D	Verificação – VER
	Validação – VAL
	Projeto e Construção do Produto – PCP
	Integração do Produto – ITP
	Desenvolvimento de Requisitos – DRE
E	Gerência de Projetos – GPR (evolução)
	Gerência de Reutilização – GRU
	Gerência de Recursos Humanos – GRH
	Definição do Processo Organizacional – DFP
	Avaliação e Melhoria do Processo Organizacional - AMP
F	Medição – MED
	Garantia da Qualidade – GQA
	Gerência de Portifólio de Projetos – GPP
	Gerência de Configuração – GCO
	Aquisição – AQU
G	Gerência de Requisitos – GRE
	Gerência de Projetos – GPR

Fonte: adaptado de SOFTEX (2009a)

Dado o contexto apresentado, nos próximos capítulos será apresentado e discutido o projeto e desenvolvimento da solução proposta para auxiliar nas atividades de GCO requeridas pelo modelo MPS.BR.

3 IDENTIFICAÇÃO DE OPORTUNIDADES E PROJETO DE UMA SOLUÇÃO PARA GERÊNCIA DE CONFIGURAÇÃO

Este capítulo descreve o contexto inicial que motivou o desenvolvimento deste trabalho, as demandas percebidas, a pesquisa por opções que pudessem suprir tais demandas e o projeto da solução implementada. Procura-se demonstrar quais foram as premissas e requisitos inicialmente levantados bem como cita-se quais as áreas definidas como fora do escopo deste trabalho.

3.1 Diagnóstico e situação inicial

Conforme descrito na seção 1.1, a motivação deste trabalho surgiu a partir da implementação do modelo MPS.BR no âmbito da empresa de desenvolvimento iProcess Soluções em Tecnologia Ltda, onde o autor atua como projetista e desenvolvedor de sistemas. A partir da implementação dos requisitos para submeter-se à avaliação formal no nível G do modelo, a empresa formalizou seu processo de desenvolvimento, definiu políticas, papéis e responsabilidades. Depois, iniciando o processo de preparação para atender ao nível F, surgiram novas demandas a serem atendidas.

Na implementação do nível G do MPS.BR, como descrito na Seção 2.3.2, a disciplina de Gerência de Configuração não é formalmente exigida (é exigida como um processo formal a partir do nível F), porém, o nível G estabelece como um dos resultados esperados do processo de Gerência de Requisitos: "GRE 3. A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida;" (SOFTEX, 2009a, p.28).

Para atender tal requisito do modelo a empresa optou inicialmente por utilizar como software de apoio para o controle da rastreabilidade o *Open Source Requirements Management Tool* (OSRMT)[1], exibido na Figura 3.1 a seguir.

[1]Ver http://sourceforge.net/projects/osrmt

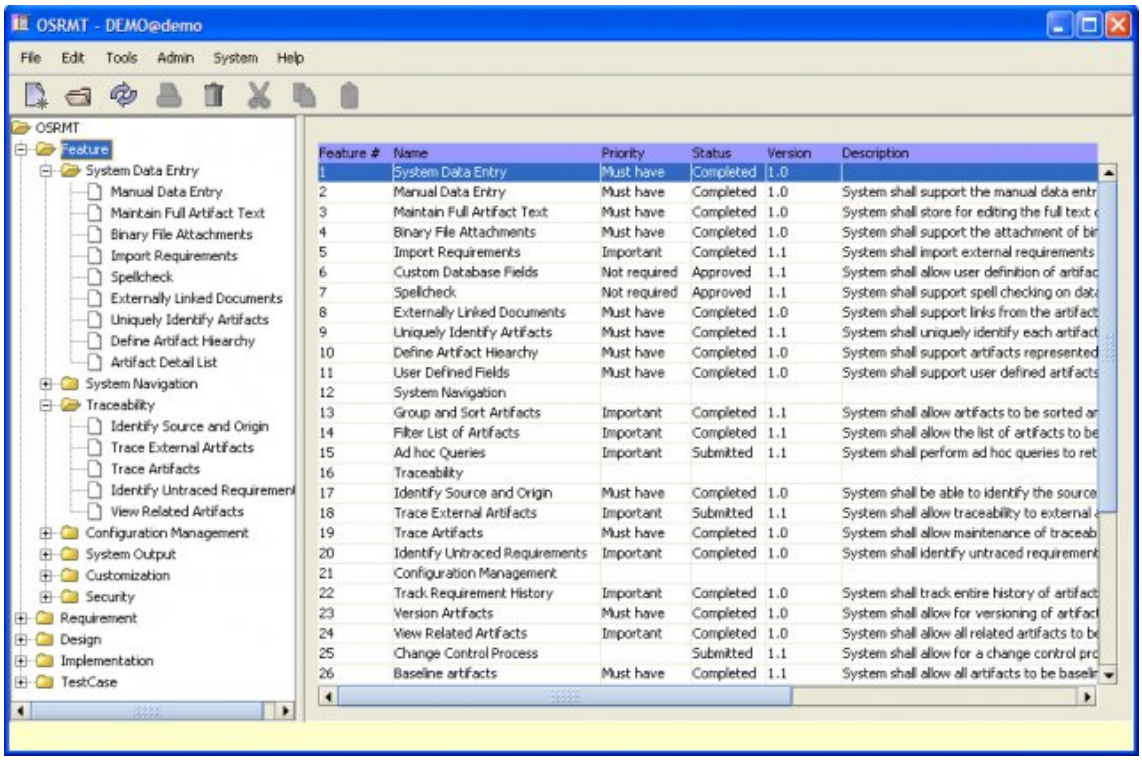

Figura 3.1: *software* de controle de rastreabilidade *OSRMT*
Fonte: http://sourceforge.net/projects/osrmt/

O uso desta ferramenta mostrou-se bastante trabalhoso. Não apenas pela dificuldade inerente à nova prática de manter bem documentados os requisitos e sua relação com os respectivos documentos de especificação funcional, modelagem, projeto e artefatos de código, mas principalmente porque a inclusão de todos estes elementos no aplicativo e a manutenção da rastreabilidade entre os mesmos deve ser realizada toda de forma manual. Na prática tem-se a percepção que o uso de uma ferramenta com tão poucas possibilidades de integração automática com os demais sistemas envolvidos no processo de desenvolvimento acaba não diferenciando-se muito do uso de formulários e planilhas pura e simplesmente. Estes por sua vez podem ser uma boa opção para testar, experimentar o controle sobre os artefatos do projeto, mas de fato não favorecem a eficiência do processo, especialmente em projetos de maior porte.

Em paralelo a esta situação, outro ponto no qual a empresa buscava evoluir era a substituição do sistema atual de controle de defeitos, *Bugzilla*[2]. Buscava-se uma ferramenta que atendesse de modo mais completo as demandas do processo de GCO que estava começando a ser formalizado visando o atendimento ao nível F do MPS.BR, especialmente nas questões de gerência de mudanças e integração com o sistema de Controle de Versões. Além disso a avaliação formal do nível G trouxe como oportunidade de melhoria a sugestão de buscar centralizar as informações relevantes do processo, que no momento estavam dispersas em diferentes planilhas, documentos e sistemas.

No decorrer desta atividade de pesquisa realizada pela empresa para buscar uma nova ferramenta de controle de defeitos, encontrou-se o software *Redmine*[3] (Figura

[2]Ver http://www.bugzilla.org/.
[3]Ver http://www.redmine.org

3.2 a seguir). Este caracteriza-se por ser uma ferramenta para gerência de projetos e gerência de *tickets*, integrada com diferentes sistemas de controle de versões (na língua inglesa *Software Configuration Management* - SCM), multilíngua e altamente personalizável. Tais características o levaram a substituir a ferramenta *Bugzilla* no controle dos defeitos dos projetos e ainda atender as demandas de gerência de mudanças.

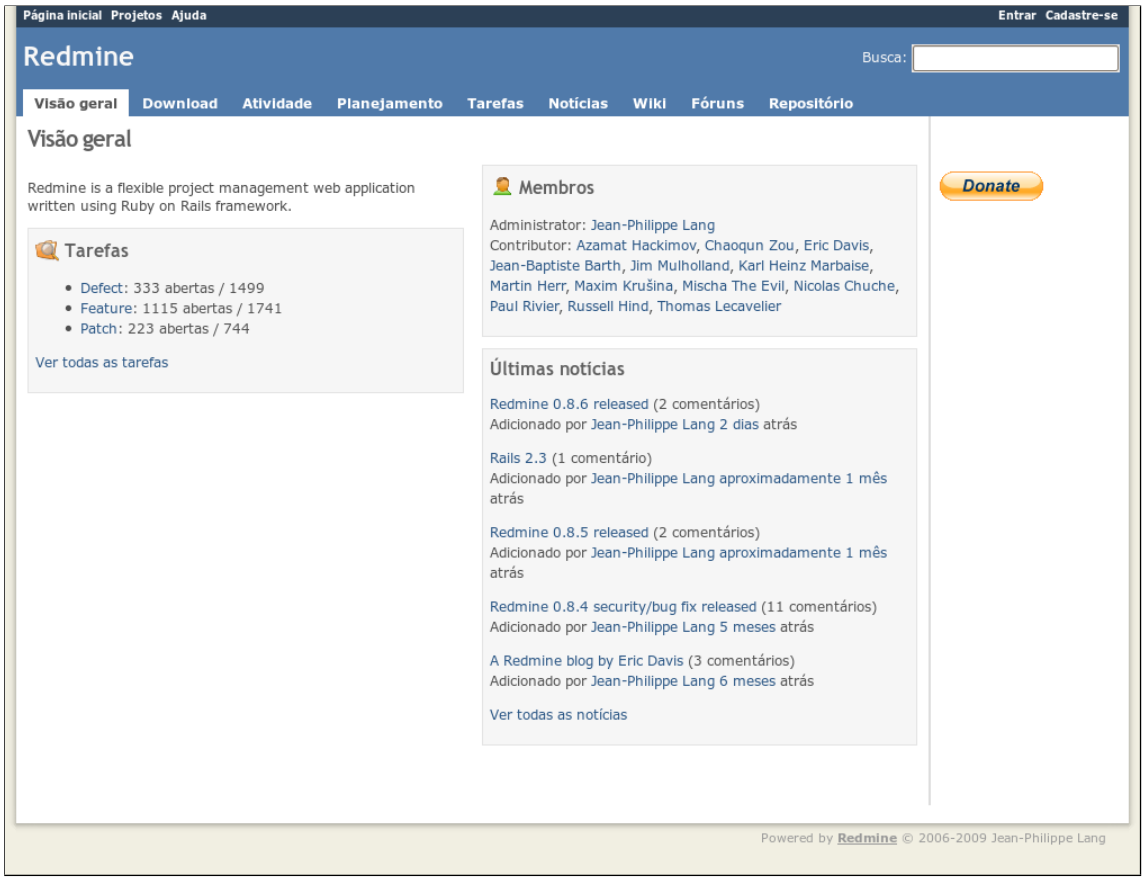

Figura 3.2: Página de visão geral da ferramenta Redmine, exibida na própria ferramenta
Fonte: http://www.redmine.org

Com o início do uso do *Redmine*, percebeu-se que ele poderia ser utilizado também para controlar a rastreabilidade dos artefatos, desde que fosse possível estender algumas de suas características e acrescentar novos módulos. Verificava-se porém que tal demanda poderia ser facilmente atendida por conta de duas características importantes desta ferramenta: primeiro, seu licenciamento é livre (*open source*), sob a *GNU General Public License v.2* (GPL), de forma que o código-fonte do sistema está disponível publicamente; segundo, sua arquitetura permite que seja facilmente expandido por meio de pequenos programas, seguindo o conceito de *plugins*[4].

Procedeu-se com o período de avaliação e teste da nova ferramenta e percebeu-se que a adaptação da equipe foi bastante fácil e rápida, de forma que a empresa optou por adotá-la de fato. Foram criados tipos de *tickets* específicos para representar os artefatos de projeto: requisitos, funcionalidades, especificações funcionais, especificações técnicas, roteiros de teste, análises de mudança, defeitos, suportes,

[4]Pequenos aplicativos desenvolvidos de forma independente do programa original e agregados ao mesmo para expandir suas funcionalidades originais. Ver http://en.wikipedia.org/wiki/Plug-in_(computing).

melhorias, planos de ação e não-conformidades. Utilizando-se a funcionalidade que a ferramenta oferece para vincular um *ticket* a outro foi possível facilmente estabelecer e manter a rastreabilidade entre estes artefatos. Fazendo uso da funcionalidade da ferramenta para integração com o repositório de controle de versões foi possível vincular os *commits* dos artefatos de código e documentos efetivamente gerados com os respectivos *tickets* no sistema, atendendo assim o resultado esperado "GRE 3" do modelo, uma vez que tem-se o relacionamento estabelecido entre os requisitos de um projeto e todos os artefatos intermediários e finais gerados a partir deste.

Para a visualização da rastreabilidade entre os requisitos e artefatos do projeto, especialmente no momento de realizar análises de impacto de mudanças, a empresa desenvolveu um *plugin* próprio. Tal desenvolvimento serviu como prova de conceito para este trabalho, demonstrando a viabilidade de estender-se o *Redmine* para implementar novas funcionalidades. Além disso, na prática diária do desenvolvimento dos projetos a empresa identificou algumas necessidades relacionadas à Gerência de Configuração, oportunidades de melhoria que deram origem à proposta deste trabalho, iniciado com o levantamento de requisitos descrito na próxima seção.

3.2 Levantamento de Requisitos

Para elicitar os requisitos deste trabalho foi realizada uma reunião de *brainstorming* tendo como tema a identificação de oportunidades de melhorias nas atividades de GCO dentro da ferramenta *Redmine*. Participaram desta reunião três dos principais integrantes do grupo de Melhoria de Processos da empresa, bem como o autor deste trabalho. Juntos estes colaboradores[5] representam diversos papéis atuantes no desenvolvimento dos projetos na empresa: gerente de projetos, analista, projetista, desenvolvedor, testador, integrador e auditor de qualidade[6].

Como resultado desta reunião foram levantados os seguintes tópicos de interesse de desenvolvimento, utilizados como insumo para a formalização dos requisitos deste trabalho:

1. Rever questões relacionadas ao Plano de Gerência de Configuração.
    - Atualmente este documento é uma planilha eletrônica;
    - É elaborado em um nível mais alto de granularidade por causa do problema ser elaborado e mantido manualmente quando detalhado no nível de artefatos do projeto;
    - Essa granularidade foi apontada como oportunidade de melhoria na avaliação da primeira meta física do nível F pela empresa;
    - Não se trabalha com um nível mais baixo de granularidade pois estima-se que iria gerar muito retrabalho manter esta informação no sistema de gerência de projetos utilizado pela empresa, nos *tickets* criados no *Redmine* e ainda por cima nessa planilha.
    - As informações deste Plano têm impacto direto nas auditorias do projeto: é neste documento que se estabelece quais artefatos fazem parte do

[5] Os nomes das pessoas não são citados por o autor considerar necessário preservar a identidade dos colegas e também porque que esta não é uma informação relevante no contexto deste trabalho.

[6] Papéis assumidos com freqüência por algum dos presentes nos projetos da empresa, em sua maioria de forma não concomitante.

projeto e qual o seu nível de configuração conforme a *baseline* em que se encontrarem. Também é estabelecido em qual *baseline* cada artefato será incluído.

2. Rever geração de *baseline*, trazendo automação do processo para dentro do *Redmine*.
   - Gerar as *baselines* de acordo com o Plano de Gerência de Configuração;
   - Uma primeira versão desta melhoria inclui automatizar o processo que hoje é feito diretamente no repositório de controle de versões para ser feito a partir do *Redmine*, copiando todo o ramo principal, independente de versões;
   - Criar um relatório automático de prováveis não-conformidades (NC), que pode verificar se as *baselines* criadas estão de acordo com o Plano de GCO.
3. Melhorar a opção de escolha de *tickets* relacionados dentro do *Redmine*.
   - Proposto por exemplo um controle *combobox* para que se escolha dentre os *tickets* do projeto;
   - Verificar a opção de criar regras de relacionamento entre os *tickets*.
4. Criar configuração da visibilidade de campos personalizados dos *tickets* conforme o status do *ticket* (útil especialmente para os *tickets* de mudança, cujos campos devem ser preenchidos aos poucos, a medida que a análise da mudança evolui).
5. Criar controle de visibilidade para clientes, restringindo a visibilidade de *tickets* de defeitos por exemplo.
6. Criar uma nova aplicação no *Redmine* para configuração do fluxo entre os status dos projetos, com a possibilidade de configurar um único fluxo para diferentes papéis (atualmente a ferramenta exige que o fluxo seja configurado para cada papel individualmente).
7. Rever os padrões de estrutura dos projetos e produtos no repositório de controle de versões e dentro do *Redmine*. Atualmente a empresa não tem uma definição clara de como lidar com produtos, suas evoluções e com projetos de personalização para clientes.
8. Tomar cuidado e garantir que todos os desenvolvimentos realizados no sistema *Redmine* sejam implementados como *plugins*, sem alteração no código-fonte original do programa, de forma a não impactar em futuras atualizações do mesmo.

Além destes, outros pontos foram discutidos e considerados não importantes para este momento, tais como:

- Integrar o *Redmine* ao sistema de gerência de projetos utilizado, *Project Builder*[7] (*software* proprietário, sem uma interface - API - para integração);
- Integrar o *Redmine* ao sistema administrativo da empresa;

[7]Ver http://www.projectbuilder.com.br

3.2.1 Itens fora do escopo

Dentre os pontos discutidos na reunião de levantamento de requisitos, alguns foram considerados como fora do escopo deste trabalho. Isso se deu por afastarem-se do tema central de Gerência de Configuração ou pela necessidade de simplificar o escopo, adequando-se à complexidade e prazo planejados do trabalho.

Os pontos considerados fora do escopo são:

- Item 3: trata-se de uma melhoria funcional, não prioritária. Tal alteração pode ser sugerida como melhoria diretamente no controle de defeitos do projeto *Redmine*.
- Item 4: não relaciona-se diretamente ao tema do trabalho.
- Item 5: tal funcionalidade já está sendo implementada pelos desenvolvedores do projeto *Redmine*, com previsão de ser incorporada na versão 0.9 da ferramenta[8].
- Item 6: não relaciona-se diretamente ao tema do trabalho.
- Item 7: este item está relacionado ao tema de GCO, mas envolve mudanças de padrões organizacionais e não alterações e desenvolvimento diretamente na ferramenta *Redmine*.

3.2.2 Definição dos requisitos

A partir do levantamento das necessidades, realizado com os usuários interessados, e da identificação de quais tópicos estão fora do escopo deste trabalho, são definidos os seguintes requisitos para o novo *plugin Redmine GCO* :

1. Criar plano de Gerência de Configuração.
2. Automatizar a geração de *baselines*, com base no plano estabelecido.
3. Criar relatório de não-conformidades.
4. Criar funcionalidade de correção da associação entre *tickets* e revisões.
5. Implementar controle de acesso às novas aplicações, por papel de uso do sistema.
6. Gerar *log* das atividades de Gerência de Configuração na ferramenta.

3.3 Pesquisa de ferramentas existentes

Uma vez elicitados os requisitos, partiu-se para a etapa de pesquisa na *internet* por ferramentas que pudessem atender às necessidades levantadas. O foco principal foi buscar *plugins* para o *Redmine*, de acordo com o objetivo deste trabalho. Porém outras ferramentas independentes que indicassem atender à demanda também poderiam ser avaliadas.

[8]Ver http://www.redmine.org/issues/337

Como resultado desta etapa, não foram encontrados *plugins* para o *Redmine* que implementassem as funcionalidades requeridas, corroborando a hipótese levantada na motivação deste trabalho (seção 1.1) quanto à inexistência de opções[9].

Como alternativa independente do *Redmine*, o *software IBM Rational ClearCase*[10] é uma das opções mais citadas na *internet* quando se fala em Gerência de Configuração, especialmente no contexto de aplicação do modelo CMMI. Porém, esta é uma ferramenta cujo licenciamento é pago, e estima-se que possua um alto custo de aquisição, fazendo parte da suíte de produtos *IBM Rational*. Além desta questão de custo de licenciamento da ferramenta, não faria sentido adotá-la dado o contexto já estabelecido na empresa iProcess de uso de *Subversion* (SVN) para controle de versões e o *Redmine*.

Outras opções comerciais também foram encontradas, como por exemplo o *software Perforce*[11]. Porém, da mesma forma como no caso da ferramenta *IBM Rational*, o uso deste sistema não seria facilmente adaptável ao contexto atual de uso do *Redmine*, de modo que a mesma não foi considerada como opção.

De um modo geral percebe-se que as ferramentas disponíveis como *software livre* são em geral específicas para determinados fins: controle de versões, gerência de mudanças ou controle de *tickets*, não havendo integração direta entre as mesmas.

Com base no resultado da pesquisa realizada atestou-se a inexistência de ferramentas que atendessem a demanda citada, demonstrando ser necessário proceder com a implementação de uma solução própria, baseada na disponibilização de um *plugin* para a ferramenta *Redmine*, integrando os aspectos de GCO com o controle de mudanças e *tickets*, e com o controle de versões dos projetos. A próxima seção trata da modelagem conceitual desta aplicação.

3.4 Modelagem do *plugin*

A partir da definição dos requisitos procedeu-se com a modelagem da solução pretendida e planejamento da implementação a ser realizada. Optou-se por utilizar a notação UML (no original *Unified Modelling Language*) para descrever o planejamento das funcionalidades no novo *plugin*, atores do sistema e sua interação com o mesmo. Porém, por se tratar de uma solução de pequeno porte, o nível de detalhamento não foi elevado, de forma a não sobrecarregar o projeto com documentação em excesso.

3.4.1 Principais atores envolvidos

Os principais atores (ou papéis) que representam os usuários do sistema, membros da equipe de desenvolvimento, no contexto das novas funcionalidades de GCO introduzidas por este *plugin* são:

- Administrador do sistema: representa o papel do responsável pela administra-

[9]Durante a revisão e redação do relatório deste trabalho o autor procurou manter-se atualizado, continuando a pesquisar na *internet* sobre o tema e ferramentas disponíveis. Assim, no final do mês de outubro de 2009 constatou-se que foi disponibilizada a versão *beta* de um *plugin* para o *Redmine* que visa atender o controle e geração de *baselines*. O mesmo está disponível em http://forge.isotrol.org/projects/show/org00006-requmngt, porém, ainda não foi avaliado pelo autor.

[10]Ver http://www-01.ibm.com/software/awdtools/clearcase

[11]Ver http://www.perforce.com

ção da ferramenta *Redmine*;

- Gerente de Projetos: representa o papel do responsável pela criação e manutenção do Plano de Gerência de Configuração, planejando e mantendo as *baselines* do projeto;
- Integrador: representa o papel do técnico responsável por criar as *baselines* do projeto;
- Auditor de Configuração: representa o papel do colaborador externo à equipe do projeto responsável por realizar as auditorias de configuração do projeto.

3.4.2 Casos de Uso

Os diagramas a seguir demonstram graficamente os casos de uso que representam as funcionalidades previstas para o novo *plugin* de GCO desenvolvido para o *Redmine* e a interação dos respectivos atores envolvidos.

É importante ressaltar que a interação dos atores demonstrada é uma configuração sugerida neste trabalho. As responsabilidades e o acesso de fato dos papéis às funcionalidades deverá ser estabelecido por meio de configurações de acesso definidas no sistema *Redmine*, permitindo que sejam definidos de forma diferente da apresentada aqui.

A Figura 3.3 a seguir exibe o caso de uso "Configurar permissões de acesso", que demonstra a possibilidade do administrador do sistema *Redmine* configurar novos papéis de uso da ferramenta e definir suas permissões de acesso. As aplicações envolvidas nestas tarefas já existem na ferramenta e este novo *plugin* deverá definir e registrar suas restrições de acesso para que sejam configuráveis por meio destas.

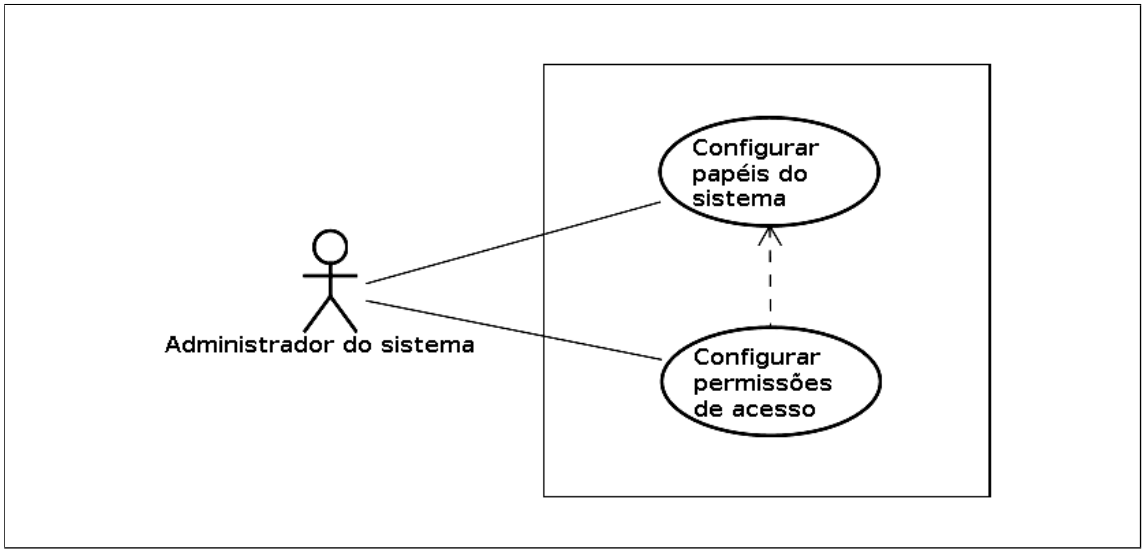

Figura 3.3: Caso de uso "Configurar permissões de acesso"

Fonte: elaborado pelo autor

A Figura 3.4 a seguir exibe o caso de uso "Editar Plano de Configuração", que demonstra a possibilidade do Gerente de Projetos manter atualizado o plano de configuração, inserindo novas *baselines* no planejamento, removendo ou alterando a ordem das *baselines* planejadas e ainda pendentes de criação.

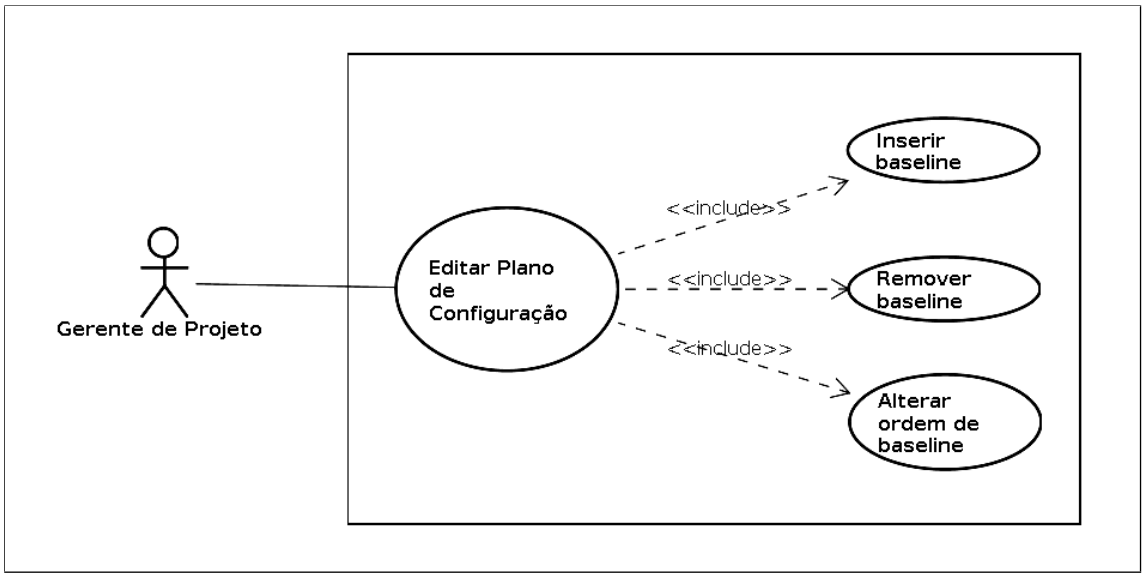

Figura 3.4: Caso de uso "Editar Plano de Configuração"

Fonte: elaborado pelo autor

A possibilidade dos atores do sistema emitirem um relatório de não-conformidades é demonstrada na Figura a seguir. Nesta, percebe-se a possibilidade do papel do Integrador do projeto alterar a associação entre *tickets* e revisões do projeto, identificados como possíveis não-conformidades no relatório.

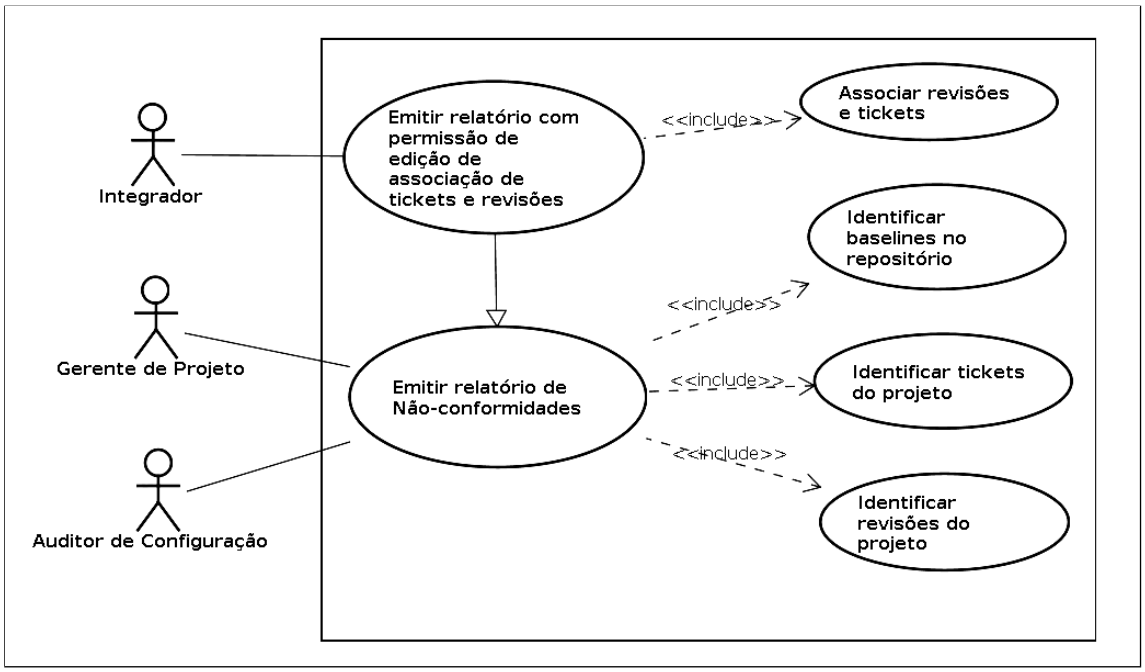

Figura 3.5: Casos de uso do relatório de não-conformidades

Fonte: elaborado pelo autor

A funcionalidade de criação de *baselines* no repositório de controle de versões é demonstrada na Figura 3.6 a seguir, no caso de uso "Criar *baseline*".

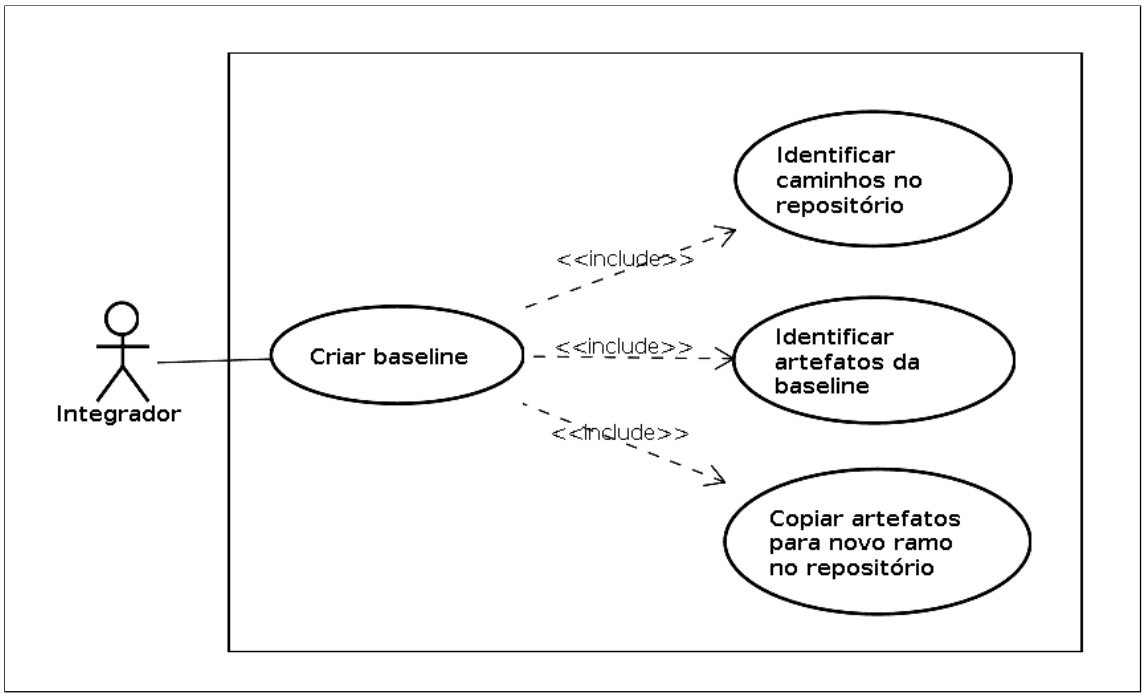

Figura 3.6: Caso de uso "Criar *baseline*"

Fonte: elaborado pelo autor

A Figura 3.7 a seguir exibe o caso de uso "Associar *tickets* e *baselines*", que estende a funcionalidade existente na ferramenta *Redmine* de edição de *tickets*. Por meio desta extensão o Gerente de Projetos indica em quais *baselines* devem ser incluídos os artefatos de cada *ticket*.

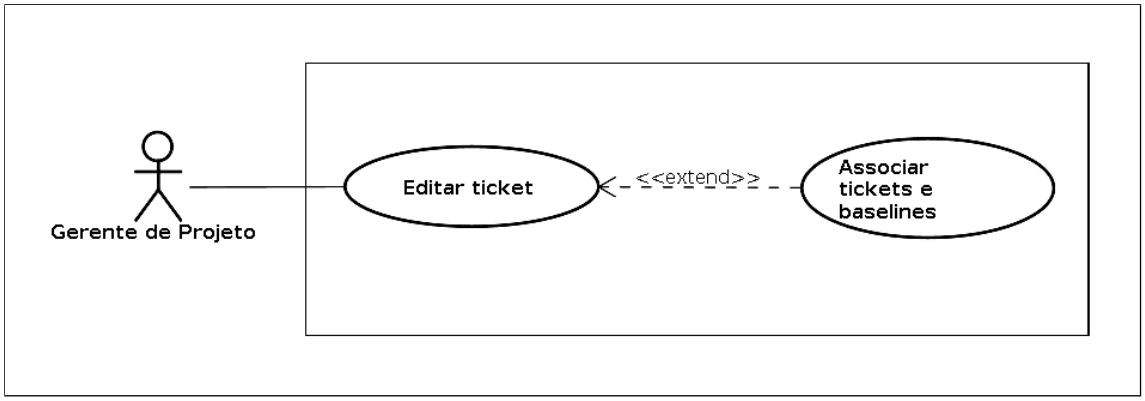

Figura 3.7: Caso de uso "Associar *tickets* e *baselines*"

Fonte: elaborado pelo autor

A Figura 3.8 a seguir exibe os casos de uso relacionados à consulta da configuração do projeto. O caso de uso "Consultar *baselines*" demonstra a funcionalidade de acesso às *baselines* já criadas para o projeto, enquanto que o caso de uso "Consultar Plano de Configuração" demonstra a funcionalidade de consulta ao Plano de Configuração estabelecido.

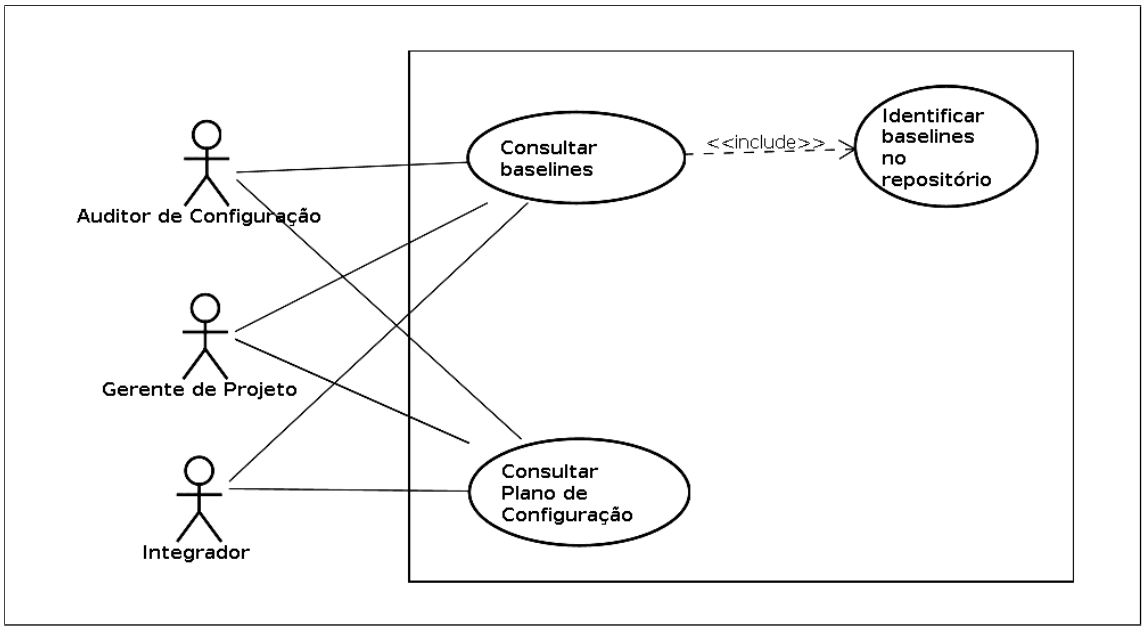

Figura 3.8: Casos de uso relacionados à consulta da configuração do projeto

Fonte: elaborado pelo autor

3.4.3 Modelo de dados

Para implementar as funcionalidades planejadas é preciso criar novos elementos no modelo de dados da ferramenta *Redmine*. Para esta primeira versão do *plugin* desenvolvido não está previsto trazer o plano de gerência de configuração por completo para dentro do sistema, mas sim o planejamento de *baselines* e o relacionamento destas com os *tickets* do projeto.

De forma a definir-se o modelo que atende tais requisitos, primeiramente analisa-se os principais conceitos do sistema e suas relações, conforme exibido na Figura a seguir. Nesta, observa-se as entidades conceituais já existentes no sistema - projetos, *tickets* e revisões - bem como a nova entidade "*baseline*" e modo como relaciona-se com as demais: um projeto possui muitos *tickets* e passa também a possuir muitas *baselines* planejadas; os *tickets* por sua vez estão associados a muitas revisões e passam também a poderem ser incluídos nas *baselines* planejadas.

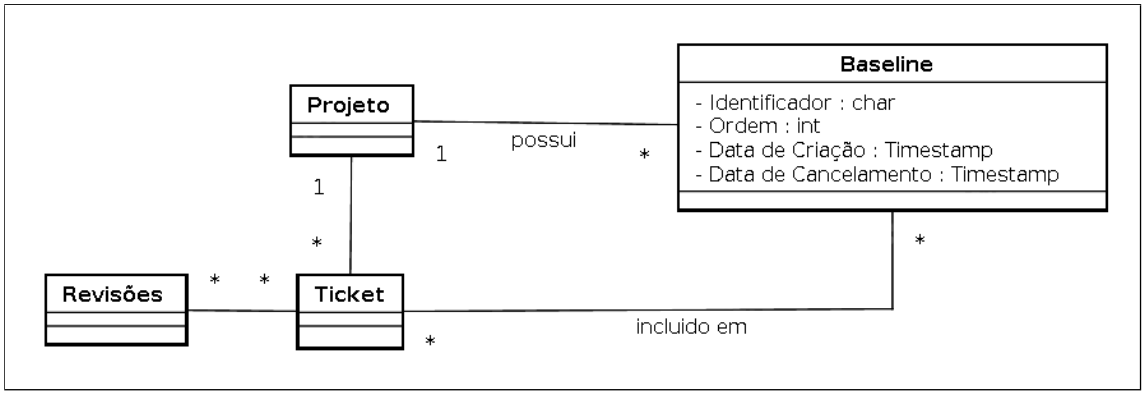

Figura 3.9: Diagrama conceitual da solução

Fonte: elaborado pelo autor

Para representar estes novos conceitos e relações, adiciona-se duas novas entidades ao modelo de dados original do *Redmine*: as tabelas "*baseline*" e "*baseline_itens*".

Estabelece-se uma nova relação “um-para-muitos” entre a entidade existente “*projects*”, responsável por representar os projetos no sistema, com a nova entidade “*baselines*”, utilizada para salvar o planejamento de *baselines* por projeto. Ainda, estabelece-se uma relação “muitos-para-muitos” entre a nova entidade “*baselines*” e a entidade “*issues*” (existente), responsável por representar os *tickets* de projetos no sistema. Esta nova relação é representada pela nova tabela “*baseline_itens*”.

O diagrama Entidade-Relacionamento (ER) exibido na Figura 3.10 demonstra em um panorama geral (sem riqueza de detalhes) a complexidade e a ordem de grandeza do modelo de dados original da ferramenta, com destaque em verde para as duas novas entidades criadas para este trabalho. A Figura 3.11 exibe com maior nível de detalhes as novas entidades (destacadas em verde) e seus relacionamentos com as entidades existentes no modelo original da ferramenta.

Ainda na Figura 3.11 observa-se a entidade “*changesets_issues*”, que representa a relação “muitos-para-muitos” entre os *tickets* de um projeto (entidade “*issues*”) e a entidade “*changesets*”, que representa as entradas (*commits*) no repositório de controle de versões. Por meio deste relacionamento é possível mapear-se a ligação entre os artefatos do projeto e os *tickets* correspondentes.

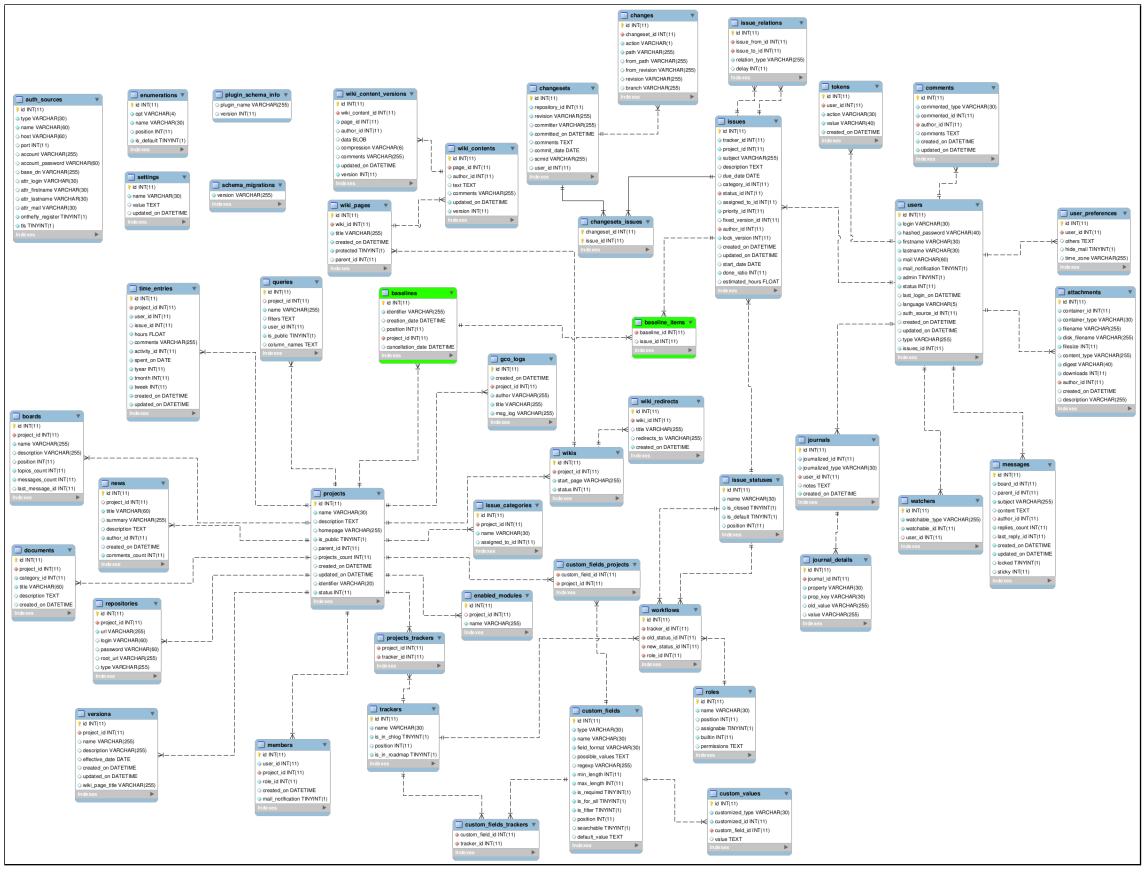

Figura 3.10: Diagrama ER da ferramenta *Redmine* - visão geral

Fonte: elaborado pelo autor

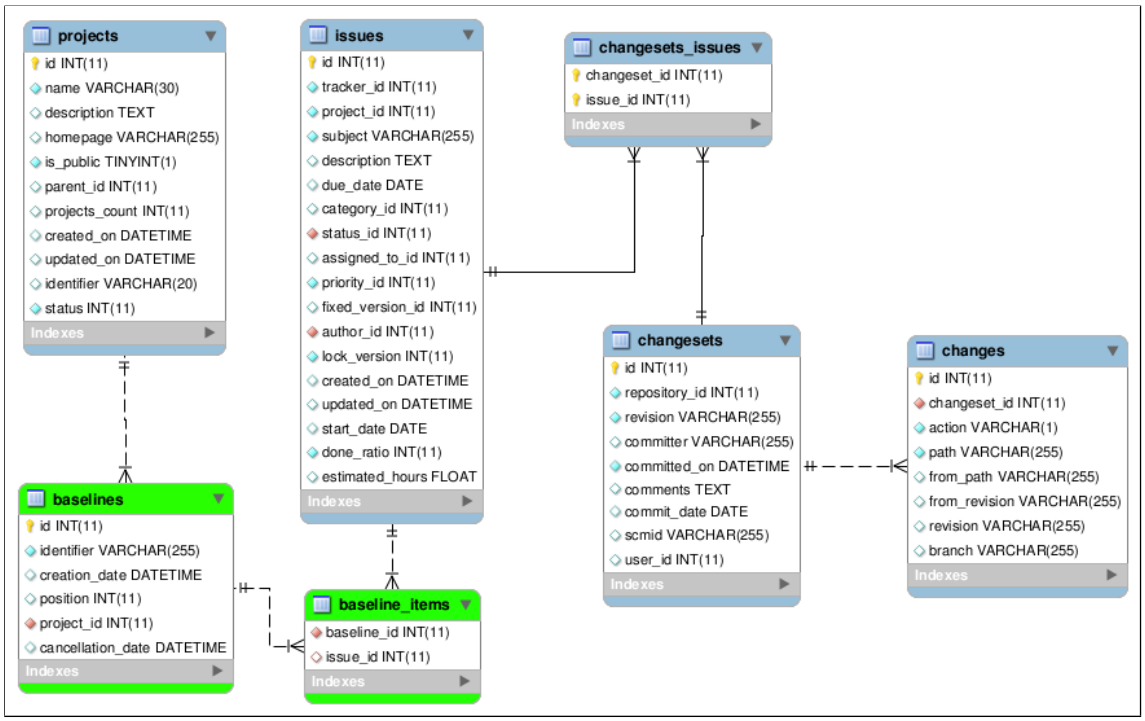

Figura 3.11: Parte do diagrama ER da ferramenta *Redmine* destacando as novas entidades e relações

Fonte: elaborado pelo autor

3.4.4 Definições de arquitetura da solução

Nesta seção destaca-se os principais aspectos da arquitetura da solução desenvolvida. Primeiramente é importante ressaltar alguns pontos acerca da própria arquitetura da ferramenta *Redmine*, uma vez que o *plugin* desenvolvido deve integrar-se com esta.

O *Redmine* possui como principais características de construção ser uma aplicação *web*, desenvolvida utilizando a linguagem de programação *Ruby*[12] e o *framework Rails*[13]. É multiplataforma, suporta diferentes Sistemas de Gerenciamento de Banco de Dados e integra-se com diferentes Sistemas de Controle de Versões.

Como decorrência direta do uso do *framework Rails* o *Redmine* segue o padrão de arquitetura *Model-View-Controller* (MVC). Este padrão consiste em "quebrar" a aplicação em basicamente três tipos de componentes: modelos (*Models*) - onde implementa-se as regras de negócio e os acesso à base de dados; leiautes (*Views*) - onde implementa-se a geração das interfaces com o usuário; e controladores (*Controllers*) - que orquestram as requisições da aplicação entre os outros componentes; conforme pode ser visto na Figura 3.12 a seguir.

[12]Ver http://www.ruby-lang.org.
[13]Ver http://rubyonrails.org/.

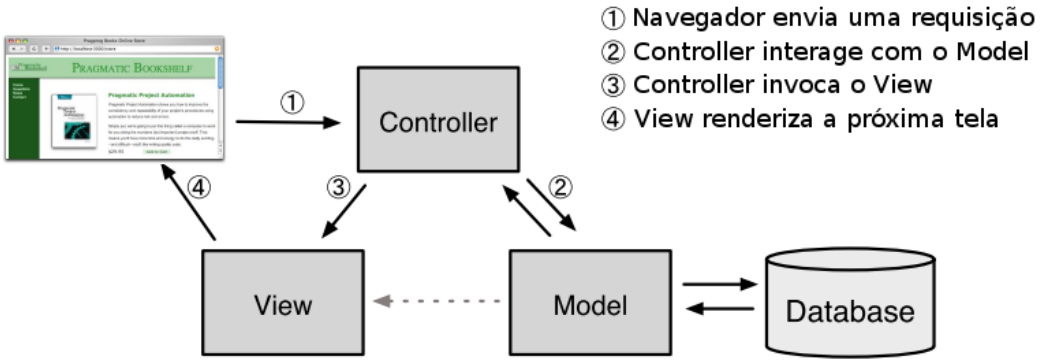

Figura 3.12: Padrão de projeto *Model-View-Controller*

Fonte: adaptado de THOMAS; HANSSON (2007, p. 23)

O *framework Rails* na verdade introduz um componente a mais na estrutura do padrão MVC que é o "roteador". Cada requisição *web* é direcionada ao roteador, que verifica o controlador responsável por tratá-la e localiza neste o método correspondente, chamado de *action*. Neste ponto, outra questão arquitetural do uso deste *framework* que teve de ser considerada na modelagem desta solução é o conceito de "convenção sobre configuração": existem diversas convenções padronizadas que, uma vez seguidas, permitem que o desenvolvedor não precise defini-las formalmente. Por exemplo, seguindo-se as convenções dos nomes das classes e métodos corretamente não é preciso definir qual deve ser o componente *View* a ser invocado por um *Controller*.

Outras características úteis inerentes do ambiente de desenvolvimento para o *Redmine* são o suporte à internacionalização das interfaces; a possibilidade de utilização de *helpers* - métodos que abstraem algum tipo de processamento de dentro dos leiautes de forma a permitir seu reuso (o próprio código da ferramenta disponibiliza diversos destes aceleradores que podem ser utilizados); o conceito de migrações da base de dados - *scripts* para realizar as alterações no modelo de dados de forma incremental; e a possibilidade de utilização de *hooks* para estender funcionalidades - métodos que podem ser implementados pelo *plugin* para, por exemplo, estender uma determinada interface da ferramenta sem precisar alterar o código-fonte principal.

Desta forma, levando-se em conta as características do ambiente de desenvolvimento disponível, definiu-se as seguintes diretrizes arquiteturais a serem seguidas na implementação deste *plugin*:

- definir os componentes seguindo o padrão MVC;
- seguir o padrão de nomenclatura de classes e métodos, de forma a evitar a necessidade de configurações adicionais;
- utilizar os *helpers* existentes quando possível;
- customizar os leiautes utilizando *Cascading Style Sheets* (CSS)[14];

[14]Ver http://www.w3.org/Style/CSS/.

Ainda, no contexto específico deste trabalho, considerando que o novo *plugin* desenvolvido deve ser baseado na infra-estrutura existente atualmente na empresa alvo e buscando diminuir a complexidade desta que é a primeira versão da solução proposta, são estabelecidas as seguintes diretrizes adicionais:

- utilizar *MySQL*[15] como gerenciador de banco de dados;
- utilizar *Subversion*[16] como sistema de controle de versões[17];
- desenvolver a interface prioritariamente na Língua Portuguesa;

[15] Ver http://www.mysql.com/.

[16] Ver http://subversion.tigris.org/.

[17] As versões específicas requeridas deste e demais componentes da solução podem ser consultadas no apêndice A deste relatório.

4 DESENVOLVIMENTO INCREMENTAL DO REDMINE GCO

Passadas as etapas iniciais de planejamento e projeto, este capítulo descreve os estágios de desenvolvimento da solução até alcançar a versão estável atual.

Na primeira seção é descrito brevemente o processo de desenvolvimento implementado. A seguir é exposta a configuração inicial dos ambientes de desenvolvimento e testes utilizados. Por fim, nas seções seguintes descreve-se os detalhes das principais versões intermediárias desenvolvidas até alcançar-se o produto atual. São descritas apenas as versões que representam mudanças significativas na evolução do projeto e, portanto, a numeração das versões apresentadas não é linear. O objetivo não é abordar as minúcias dos algoritmos desenvolvidos mas sim relatar as principais dificuldades encontradas e decisões tomadas que guiaram a implementação até o estágio em que se encontra. Cópias das telas de cada versão sendo executada no ambiente de testes são incluídas de forma a demonstrar a evolução alcançada.

4.1 Descrição do processo de desenvolvimento

É importante ressaltar a metodologia de trabalho utilizada para o desenvolvimento, onde procurou-se desenvolver de modo ágil, realizando iterações curtas, agregando funcionalidades e gerando valor a cada nova versão. Desta forma possibilitou-se ter sempre versões funcionais, de alguma forma úteis ainda que, nas primeiras versões, bastante limitadas em relação às funcionalidades planejadas.

Cada nova versão foi instalada primeiramente em um ambiente de testes local, sendo validada pelo próprio autor utilizando-se dados reais de projetos da empresa iProcess. A seguir, a versão era instalada em um ambiente de homologação, acessível aos usuários interessados no projeto. A estes, fornecia-se apenas instruções básicas de acesso à ferramenta, sem maiores detalhes do modo de uso esperado, de forma a permitir que realizassem testes sem interferências. O *feedback* dos testes realizados pelos usuários era reportado diretamente em conversas no ambiente de trabalho e, em alguns casos, por troca de *e-mails*. Os erros reportados e as sugestões de melhorias eram assimilados no decorrer do desenvolvimento da versão seguinte.

Em termos de escala dos testes realizados com dados reais da empresa iProcess, estima-se que o sistema possua capacidade de suportar volumes de trabalho muito maiores que os testados, que foram:

- ambiente com 10 projetos existentes;
- criação de até 10 *baselines* por projeto;

- inclusão de até 15 baselines no plano de Gerência de Configuração;
- utilizados projetos com até aproximadamente:
  - 200 *tickets*;
  - 300 revisões associadas;
  - 200 não-conformidades identificadas.

4.2 Configuração inicial dos ambientes de desenvolvimento e testes

Como primeira etapa do desenvolvimento deste trabalho se estabeleceu uma infra-estrutura de ambientes de forma a dar o suporte necessário à implementação, testes, homologação e gerência de configuração do próprio projeto. Os principais componentes desta arquitetura são descritos a seguir.

Sistema de Controle de Versões

Para manter o controle de versões tanto do código-fonte desenvolvido como dos documentos relacionados ao projeto, utilizou-se o sistema de controle de versões *Subversion*, mantendo-se o repositório hospedado em um servidor externo pré-contratado e trabalhando-se com cópia de trabalho na máquina local do autor.

Ambiente de desenvolvimento

O ambiente de desenvolvimento utilizado consiste em um sistema *Linux 32 bits*, *NetBeans*[1] como ambiente integrado de desenvolvimento (IDE no original em inglês) para a linguagem *Ruby*, base de dados *MySql*, *WebBrick* como servidor *HTTP* (nativo do *framework Rails*) e *Redmine*.

Ambiente de testes

Como ambiente de testes utilizou-se uma cópia da máquina virtual com o sistema *Linux 32 bits* em uso no ambiente de produção da empresa iProcess. Desta forma possibilitou-se realizar testes com projetos na ferramenta *Redmine* com maior volume de dados. Além disso validou-se a implementação no contexto real onde a solução deverá ser executada futuramente.

Ambiente de homologação

Como ambiente de homologação, disponível para testes dos demais interessados neste projeto, utilizou-se uma instalação do *Redmine* no mesmo servidor externo onde configurou-se o repositório de controle de versões.

4.3 Versões implementadas

4.3.1 Versão 0.1

A primeira versão desenvolvida do plugin *Redmine GCO*, exibida na Figura 4.1 a seguir, teve como objetivo principal atender de forma simples a geração de *baselines*

[1] Ver http://netbeans.org/.

no repositório do projeto. Para tanto, primeiramente foram desenvolvidas algumas características básicas tais como: configurar o *plugin* para ser exibido no *menu* de abas do projeto ativo; implementar um modo de identificar qual o caminho no repositório de versões do projeto que representa o ramo principal e qual o caminho onde devem ser criadas as *baselines*, uma vez que os nomes destes caminhos (e a estrutura da árvore do repositório de um modo geral) não são necessariamente padronizados.

Buscando a simplicidade e aproveitamento das funcionalidades que o *Redmine* oferece, optou-se por utilizar a aplicação de criação de campos personalizados por projeto, disponível na ferramenta. Desta forma foram criados dois novos campos para representarem os caminhos no repositório do ramo principal e do ramo de *baselines* (ver Figura 4.2, campos "Ramo principal" e "Caminho para baselines" respectivamente). Para o acesso ao repositório verificou-se o código que o *Redmine* implementa para fazer a busca das versões vinculadas ao projeto. Este faz uso diretamente do comando do cliente *Subversion* (SVN) instalado no sistema operacional, não suportando tratamento de exceções ou de mensagens de retorno.

Como outras características importantes desta versão temos: a geração de baselines é realizada copiando-se (comando "*svn copy*") todo o conteúdo do ramo principal do repositório, na revisão mais atual; não são impostas restrições de acesso, de forma que todos os integrantes do projeto têm acesso à criação de *baselines*; para registrar o histórico corretamente a execução do comando no repositório é realizada solicitando-se o *login* e senha do usuário ao invés de utilizar-se as configurações de acesso salvas nos dados do projeto.

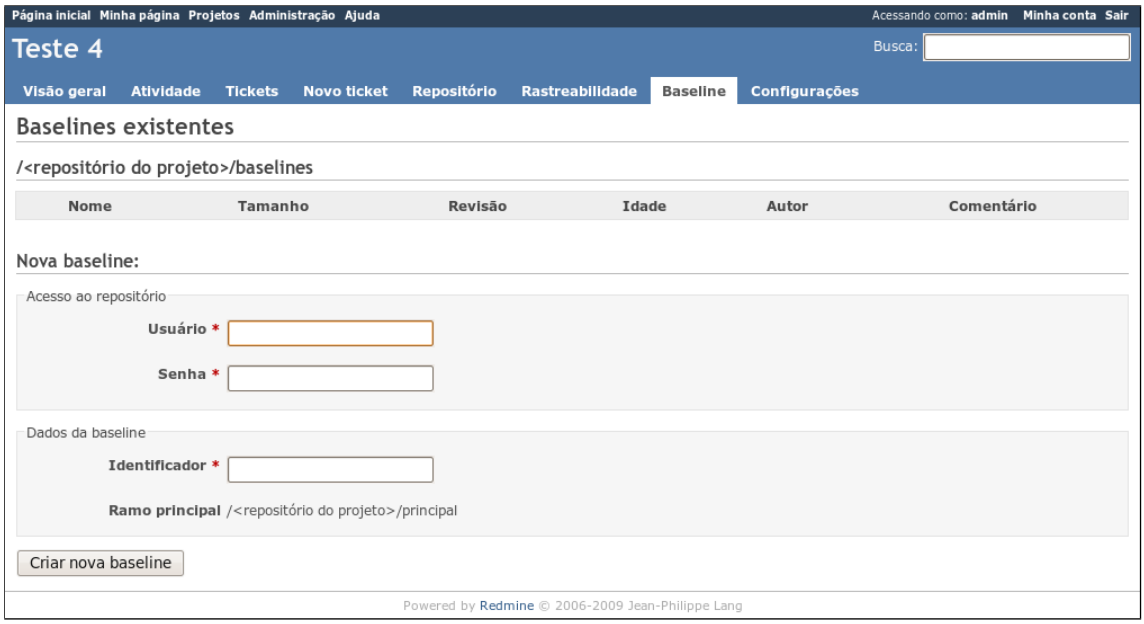

Figura 4.1: *Redmine GCO*, versão 0.1 - tela principal

Fonte: elaborado pelo autor

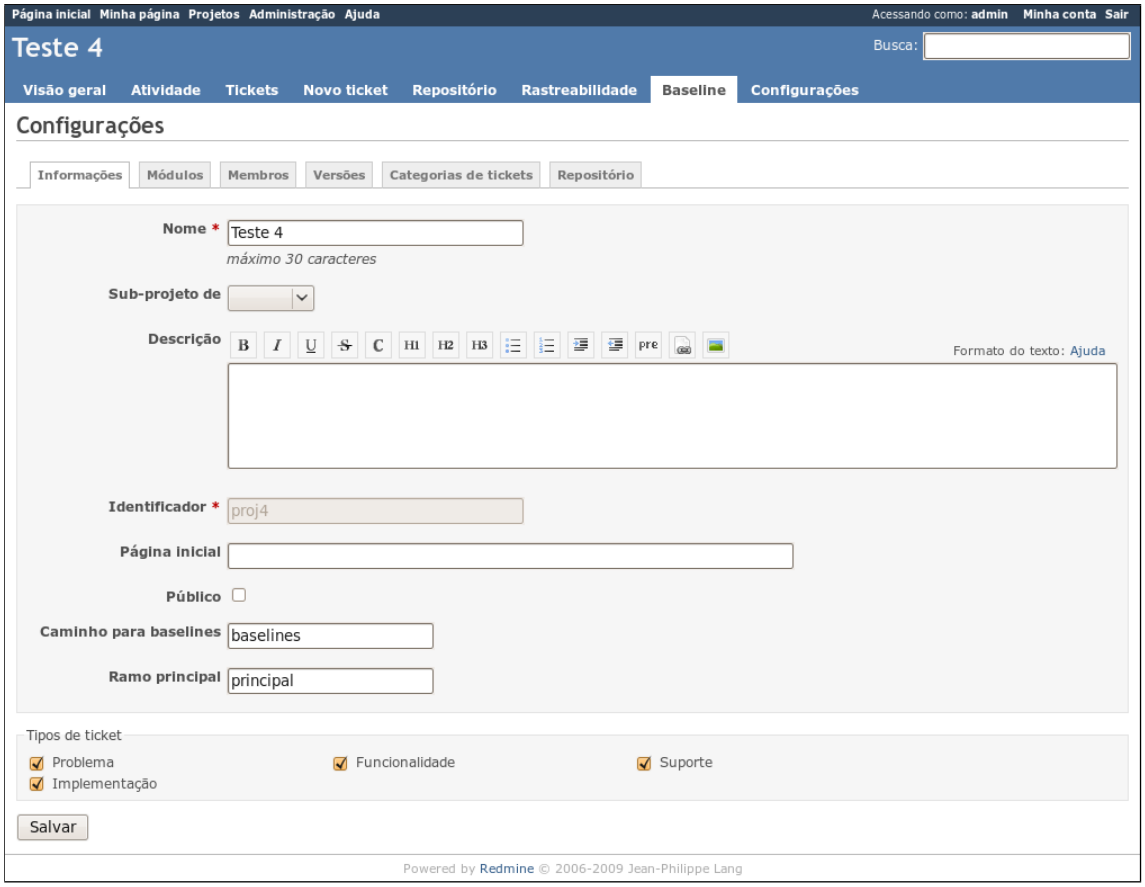

Figura 4.2: *Redmine GCO*, versão 0.1 - novos campos de dados do projeto

Fonte: elaborado pelo autor

Esta versão 0.1 foi utilizada como uma "prova de conceito" inicial, de forma a validar a viabilidade da solução proposta, bem como familiarizar o autor com o estilo e as características da programação em *Ruby on Rails* especialmente dentro do *Redmine*. Mesmo assim, seguindo as premissas definidas para o modelo de processo de desenvolvimento adotado, esta é uma versão funcional, que permite a criação de *baselines* e posterior consulta de dentro da ferramenta.

4.3.2 Versão 0.2

A segunda versão do *plugin* desenvolvido representa o fechamento da fase inicial de ambientação com as tecnologias envolvidas e o início de fato do caminho percorrido para alcançar-se a versão atual. A primeira mudança em relação à versão anterior é a alteração da interface principal da aplicação: é alterado o nome da aba no menu do projeto de "Baseline" para "GCO", e a tela principal da aplicação passa a ser subdividida em seu próprio menu de abas, conforme exibido na Figura 4.3 a seguir. Desta forma se estabelece um ponto central na ferramenta para concentrar as novas funcionalidades relacionadas à Gerência de Configuração.

Figura 4.3: *Redmine GCO*, versão 0.2 - tela principal

Fonte: elaborado pelo autor

A aba "Baselines" da tela principal exibida na Figura 4.3 carrega a mesma aplicação desenvolvida na versão 0.1, apenas com alguns ajustes visuais. A aba "Plano de Configuração" carrega a nova aplicação para criação do planejamento das *baselines* do projeto, exibida na Figura 4.4.

Figura 4.4: *Redmine GCO*, versão 0.2 - plano de configuração

Fonte: elaborado pelo autor

Uma vez criada por meio desta ferramenta uma *baseline* cujo identificador tenha sido previamente planejado, fica registrada a sua data de criação no Plano de Configuração e sua ordem não pode mais ser alterada. As demais *baselines* planejadas e ainda pendentes ficam passíveis de serem reordenadas.

4.3.3 Versão 0.5

Esta versão apresenta primeiramente melhorias de usabilidade da aplicação. A criação de *baselines* passa a apresentar uma caixa de seleção (*combobox*) para que se escolha o identificador da *baseline* a ser criada a partir do Plano de Configuração. Desta forma impede-se a utilização de identificadores não previstos, assegurando-se a padronização da nomenclatura. A aplicação de "Plano de Configuração" por sua vez passa a suportar a exclusão dos itens planejados, que ainda não havia sido implementada até a versão anteriormente descrita.

É também nesta versão que disponibiliza-se a nova aplicação de relatório de possíveis não-conformidades, exibido na Figura 4.5. Esta aplicação serve de guia

para uma auditoria de configuração do projeto, explicitando neste momento dois focos de possíveis situações indesejadas: a existência de *baselines* no repositório mas que não constam no Plano de Gerência de Configuração; e a existência de *baselines* no repositório que foram devidamente planejadas mas que ainda constam no sistema como pendentes. Ambas as situações indicam que o repositório foi manipulado externamente ao *Redmine*, e tal situação deve ser apurada.

Figura 4.5: *Redmine GCO*, versão 0.5 - relatório de não-conformidades

Fonte: elaborado pelo autor

4.3.4 Versão 0.8

A oitava versão do *plugin Redmine GCO* apresenta algumas mudanças importantes nas funcionalidades da aplicação, especialmente no modo como o repositório é manipulado.

Primeiramente o relatório de não-conformidades foi complementado com mais uma seção: "Revisões no ramo principal não associadas a nenhum *ticket*". A verificação deste tipo de situação é importante visando-se manter correta a rastreabilidade entre os produtos de trabalho que, no *Redmine*, é mantida por meio do relacionamento entre as revisões no repositório e os *tickets* no sistema. Se uma revisão não possui em seu comentário a indicação do número do *ticket* que a originou, os arquivos modificados por esta revisão não aparecem na rastreabilidade do projeto. Não tendo sido incluídos no relatório de rastreabilidade é mais difícil para o auditor de configuração verificar em qual *baseline* estes arquivos deveriam estar presentes. Ainda, de forma a minimizar estes problemas e facilitar o trabalho de auditoria, foi criada a possibilidade de se corrigir a não-conformidade, indicando qual o ticket ao qual a revisão listada deveria estar relacionada, conforme exemplificado na Figura 4.6 a seguir.

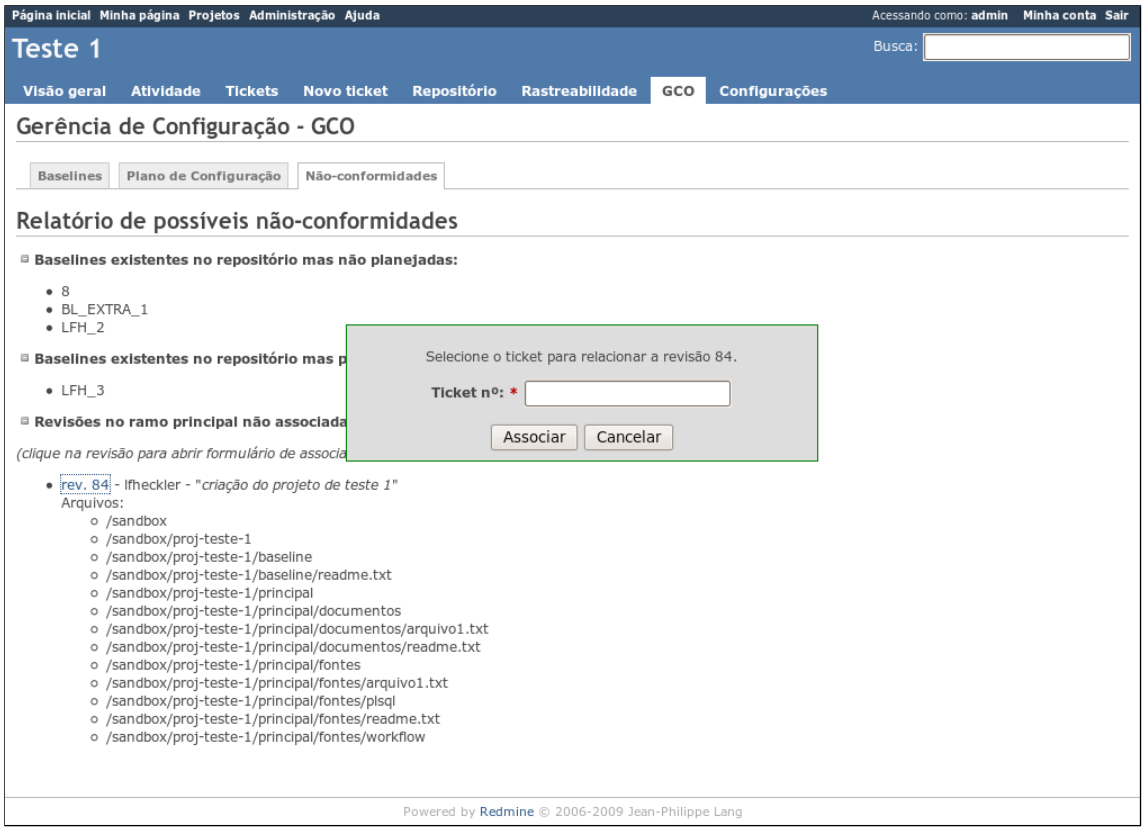

Figura 4.6: *Redmine GCO*, versão 0.8 - correção de não-conformidade

Fonte: elaborado pelo autor

Outras mudanças importantes foram realizadas na funcionalidade de criação das *baselines*. Os defeitos reportados nos testes das versões anteriores foram corrigidos e foi incluído o conceito de cancelamento automático das *baselines* planejadas e não criadas. Por meio desta funcionalidade, quando o usuário seleciona para ser criada uma *baseline* que não seja a imediatamente próxima no planejamento, as anteriores pendentes são automaticamente canceladas. Desta forma é possível manter a informação de qual foi o planejamento inicialmente previsto sem, no entanto, engessar o processo. A Figura 4.7 apresenta o plano de configuração para um projeto fictício, demonstrando a série de *baselines* que foram efetivamente criadas no projeto e as que foram canceladas automaticamente.

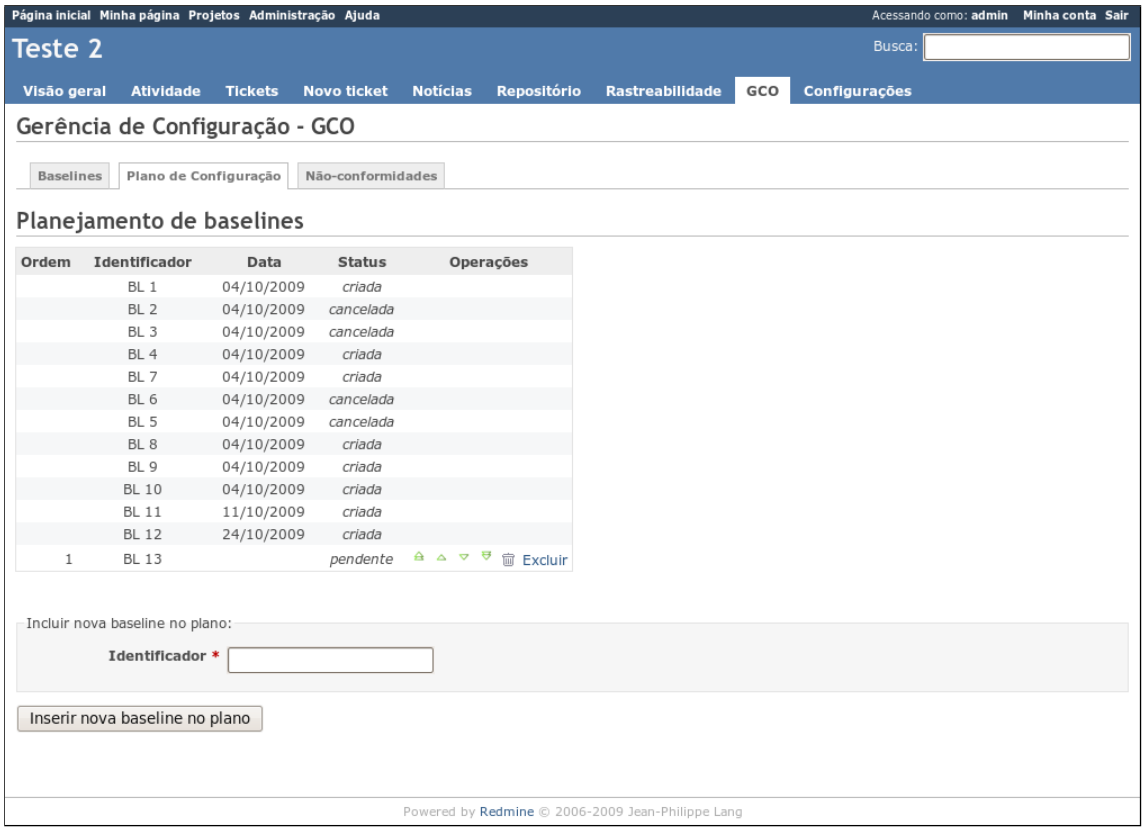

Figura 4.7: *Redmine GCO*, versão 0.8 - cancelamento automático de *baselines*

Fonte: elaborado pelo autor

Outro aspecto que evoluiu nesta versão foi a questão da seleção automática dos arquivos no ramo principal que devem fazer parte de cada *baseline*. Para esta questão identificou-se duas abordagens possíveis:

1. Copiar o ramo principal por completo, na sua revisão mais atual (opção atualmente implementada);

2. Copiar apenas os arquivos do ramo principal cujas revisões estejam associadas à *baseline* que estiver sendo criada.

A abordagem 1, atualmente implementada, implica em uma mudança no processo de desenvolvimento, onde a equipe deve ter maior atenção para manter no ramo principal apenas os produtos de trabalho que de fato devam ser incluídos na próxima *baseline*. Isso implica por exemplo que, supondo que o time de desenvolvimento esteja dividido e que parte da equipe está adiantando trabalho da iteração seguinte, estes devem trabalhar em um ramo paralelo (*branch*) que deverá ser consolidado com o ramo principal apenas depois de criada a *baseline* da iteração atual. Desta forma, pode ocorrer de serem copiados artefatos indevidamente, apenas por estarem presentes no ramo principal.

Já a abordagem 2 permite que se tenha no ramo principal produtos de trabalho que devem ser incluídos em *baselines* diferentes ou mesmo produtos cujas revisões anteriores é que devem ser copiadas ao criar-se a nova *baseline* no repositório. Porém, esta opção exige que se tenha um controle mais rigoroso na manutenção do relacionamento entre as revisões do repositório e as *baselines*, sob o risco de algum

artefato não ser copiado por não estar corretamente relacionado, ainda que esteja presente no ramo principal.

Como este trabalho tem por foco atender primeiramente as demandas da empresa alvo do estudo, os fornecedores de requisitos da empresa foram consultados. A partir das discussões realizadas decidiu-se que a opção 2 seria a abordagem mais indicada para o contexto atual da empresa. Desta forma, partiu-se para a implementação do módulo de relacionamento entre as *baselines* planejadas e os *tickets* do projeto, e a refatoração da funcionalidade principal de geração de *baselines* para selecionar apenas os arquivos nas revisões esperadas.

O desenvolvimento do módulo de relacionamento entre *tickets* e *baselines* foi realizado aproveitando que o *Redmine* disponibiliza um ponto de extensão da tela de consulta de *tickets*. Por meio deste *hook* foi implementada uma classe que gera o leiaute da nova funcionalidade, inserido na tela de consulta de um *ticket* quando esta é acessada (sem a necessidade de alterar-se o código-fonte original do *Redmine*). A Figura 4.8 a seguir demonstra a exibição da nova seção, que lista todas as *baselines* do projeto, explicita aquelas que já foram criadas e incluíram este *ticket* e permite associá-lo com aquelas que ainda estejam pendentes no planejamento.

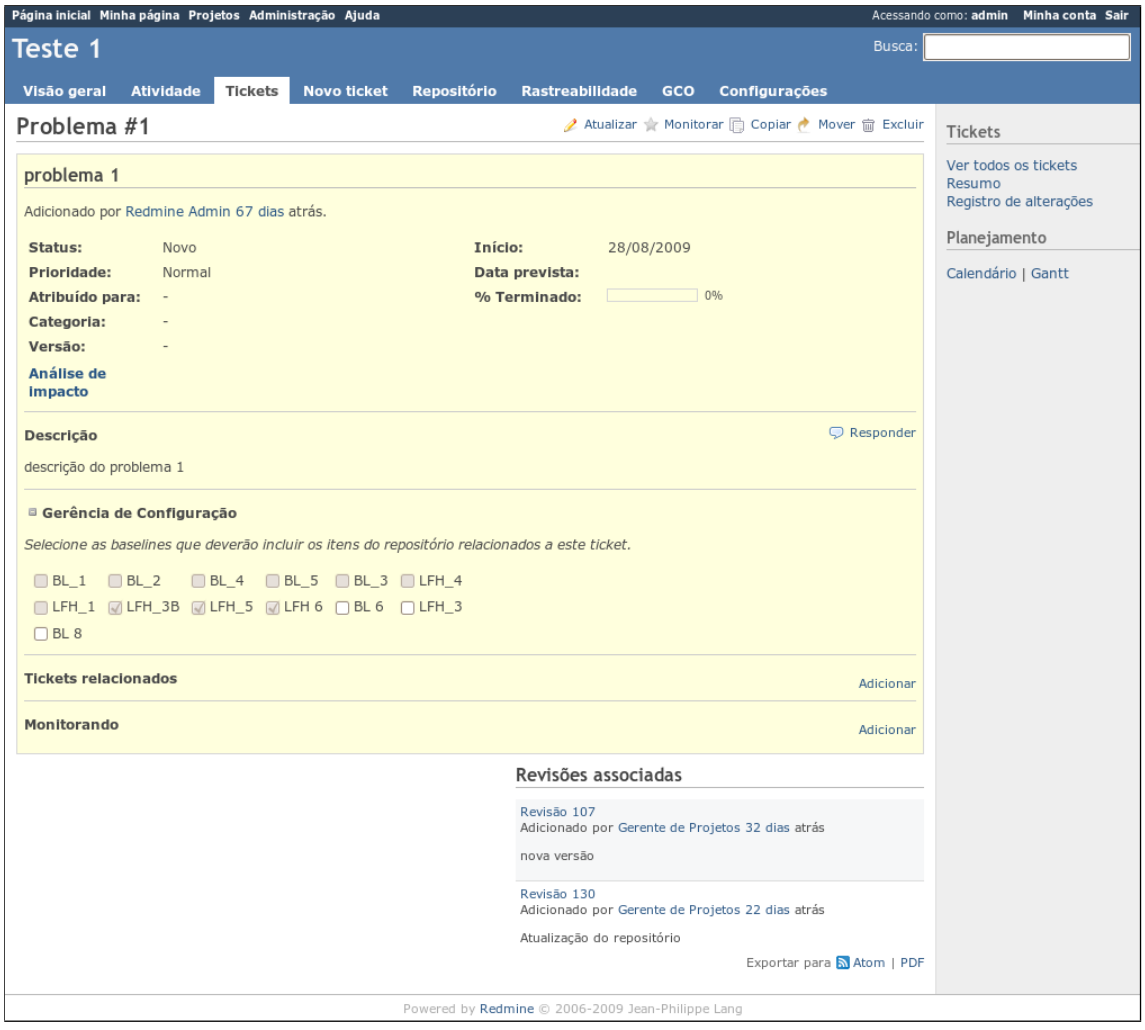

Figura 4.8: *Redmine GCO*, versão 0.8 - associação de *tickets* e *baselines*

Fonte: elaborado pelo autor

Para a implementação da criação da *baseline* incluindo apenas os artefatos nas

revisões esperadas, encontrou-se dificuldade no modo como o repositório *Subversion* é acessado: a implementação atual da conexão do *Redmine* faz uso diretamente do cliente SVN do servidor, sem suporte a tratamento de exceções. Desta forma, no desenvolvimento das versões intermediárias a esta o código havia sido alterado para utilizar a biblioteca *SWIG*[2] para conexão com SVN utilizando-se a linguagem *Ruby* por meio de *wrappers* entre esta linguagem e a API do cliente *Subversion* nas linguagens C e C++. O uso desta biblioteca proporcionou implementar-se tratamento de exceções na criação das *baselines* tratando por exemplo erros no usuário e senha informados, servidor indisponível, caminhos inválidos, etc. Porém, a versão disponível da biblioteca para a linguagem *Ruby* não possui suporte para o comando de cópia selecionando por revisão, o que é requerido para implementar a nova diretriz de seleção dos artefatos. Desta forma, optou-se por voltar a utilizar o cliente SVN diretamente para a execução dos comandos, abrindo mão temporariamente do tratamento de exceções. A Figura 4.9 a seguir demonstra a nova aplicação de criação de baselines, destacando-se a nova funcionalidade: ao ser selecionado o identificador da *baseline* desejada na caixa de seleção, por meio de uma requisição *AJAX* retorna-se a lista dos artefatos cujas revisões estão associadas à *baseline* selecionada.

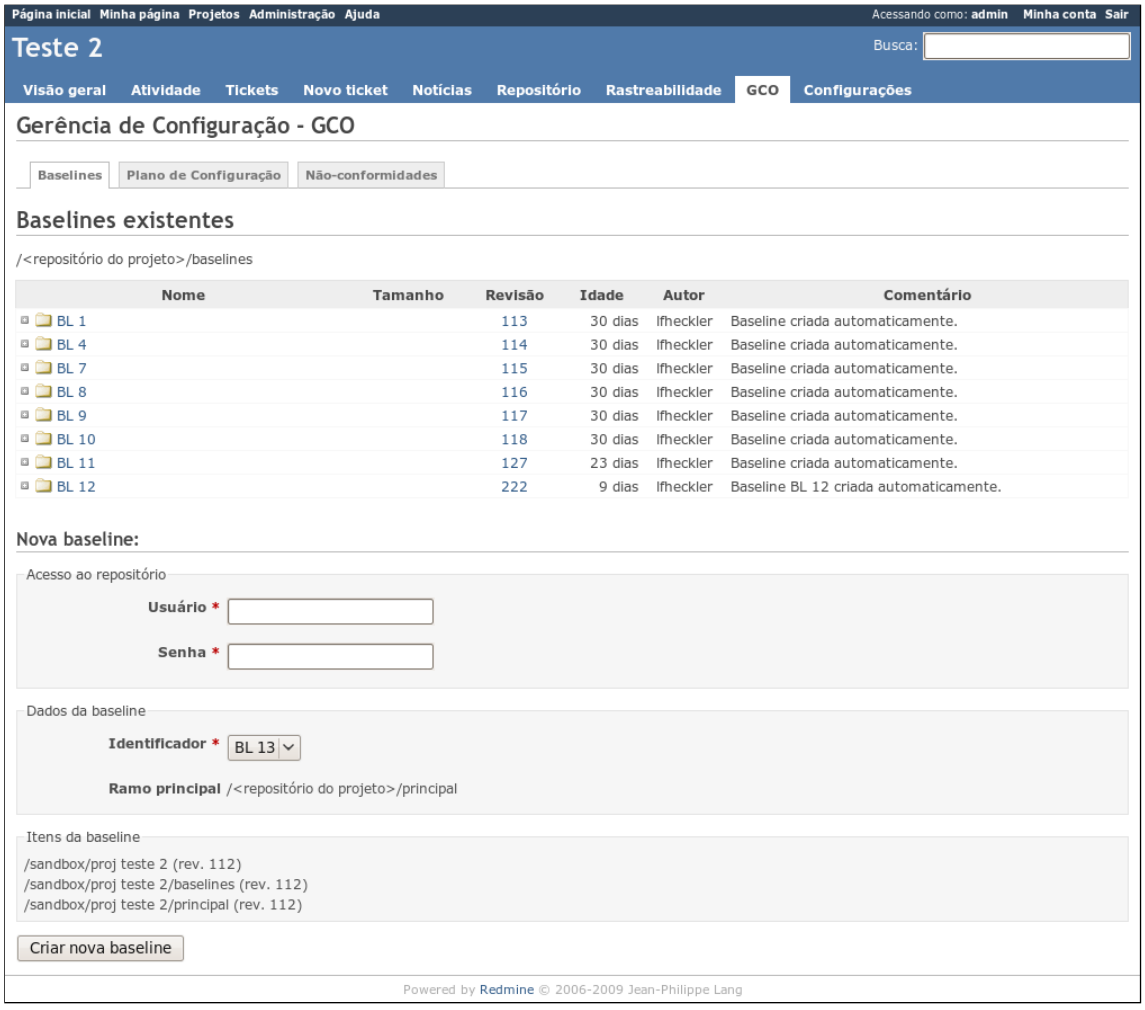

Figura 4.9: *Redmine GCO*, versão 0.8 - seleção de artefatos da *baseline*

Fonte: elaborado pelo autor

[2]Ver http://www.swig.org/.

4.3.5 Versão 1.0

A versão 1.0 corresponde à 12ª versão de desenvolvimento do *plugin* e consiste na versão estável que considera-se que atende os objetivos deste trabalho. Como principais evoluções desta versão temos novamente o suporte ao tratamento de exceções, correção de defeitos, possibilidade de escolha dos artefatos que compõem a *baseline*, melhorias na associação de *tickets* e revisões, registro de histórico das ações realizadas e configuração de permissões de acesso por perfil.

O suporte ao tratamento de exceções foi implementado utilizando-se uma nova biblioteca para execução de comandos do sistema operacional na linguagem *Ruby*: *systemu*[3]. Este módulo permite não apenas testar-se o retorno com sucesso ou não da execução, mas acessar mensagens de retorno ou de exceção. A Figura 4.10 a seguir ilustra o tratamento de exceção para o caso de *login* inválido no repositório.

Figura 4.10: *Redmine GCO*, versão 1.0 - tratamento de exceções

Fonte: elaborado pelo autor

Quanto à criação de *baselines*, verifica-se que uma revisão no repositório pode compor artefatos que não deveriam ser incluídos na *baseline* selecionada e outros que sim, e a ação de *commit* do artefato na revisão não tem como ser desfeita. Por exemplo, o *commit* da revisão pode incluir artefatos que não se encontram no ramo principal e portanto não devem ser incluídos na geração da *baseline*. Desta forma implementou-se tratamento desta situação específica e disponibilizou-se ao usuário a possibilidade de desmarcar determinados artefatos para que não sejam incluídos na *baseline* selecionada, conforme verifica-se na Figura 4.11 a seguir.

[3]Ver http://codeforpeople.rubyforge.org/svn/systemu/trunk/README.

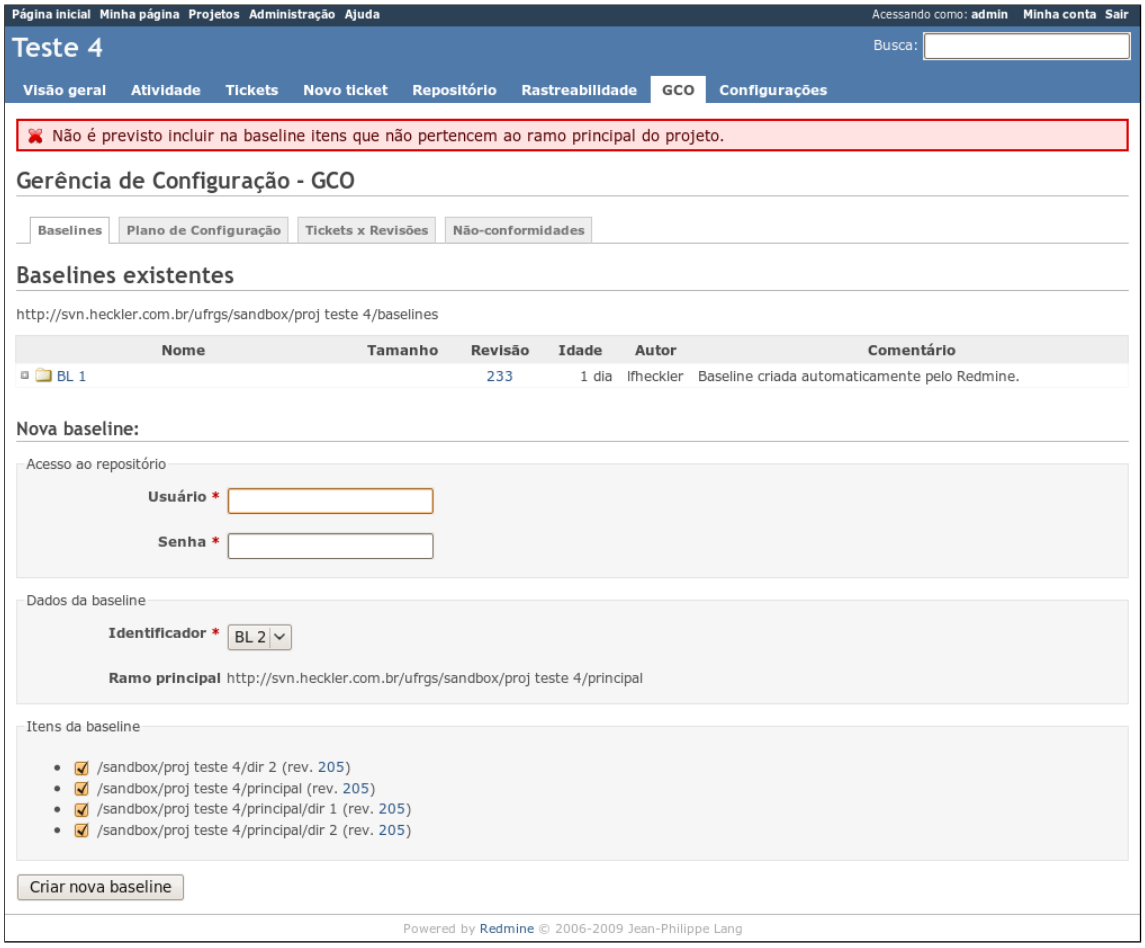

Figura 4.11: *Redmine GCO*, versão 1.0 - seleção de artefatos que compõem a *baseline*

Fonte: elaborado pelo autor

Figura 4.12: Redmine GCO, versão 1.0 - módulo de associação de *tickets* e revisões

Fonte: elaborado pelo autor

Ainda, temos que a funcionalidade de associação de revisões e *tickets* recebeu

duas evoluções. Primeiramente a seleção de *tickets* diretamente no relatório de não-conformidades (Figura 4.6 da seção 4.3.4) foi atualizada para que o usuário possa selecionar o *ticket* em uma caixa de seleção, dentre os *tickets* do projeto. Esta abordagem traz duas vantagens principais: impede o usuário de associar a revisão a um *ticket* que não seja do projeto; e permite que ele selecione o *ticket* ao invés de ser obrigado a digitar o código manualmente. Porém, isso não impede que o usuário cometa um engano e associe a revisão erroneamente, ação esta que não havia como ser desfeita até então. Desta forma, a segunda evolução desenvolvida foi a implementação de um módulo específico para associação entre os *tickets* e revisões, demonstrado na Figura 4.12 anterior.

Outro ponto importante desenvolvido nesta versão do *plugin* é a geração de histórico das ações de gerência de configuração. Esta funcionalidade é importante para que se possa rastrear quais foram as ações tomadas caso proceda-se uma auditoria, ou caso seja preciso desfazer alguma ação realizada erroneamente. Para tanto, as ações de associação entre *tickets*, *baselines* e revisões, bem como o planejamento e a criação das *baselines* geram entradas no *log* de atividades do projeto, conforme exemplificado na Figura 4.13 a seguir.

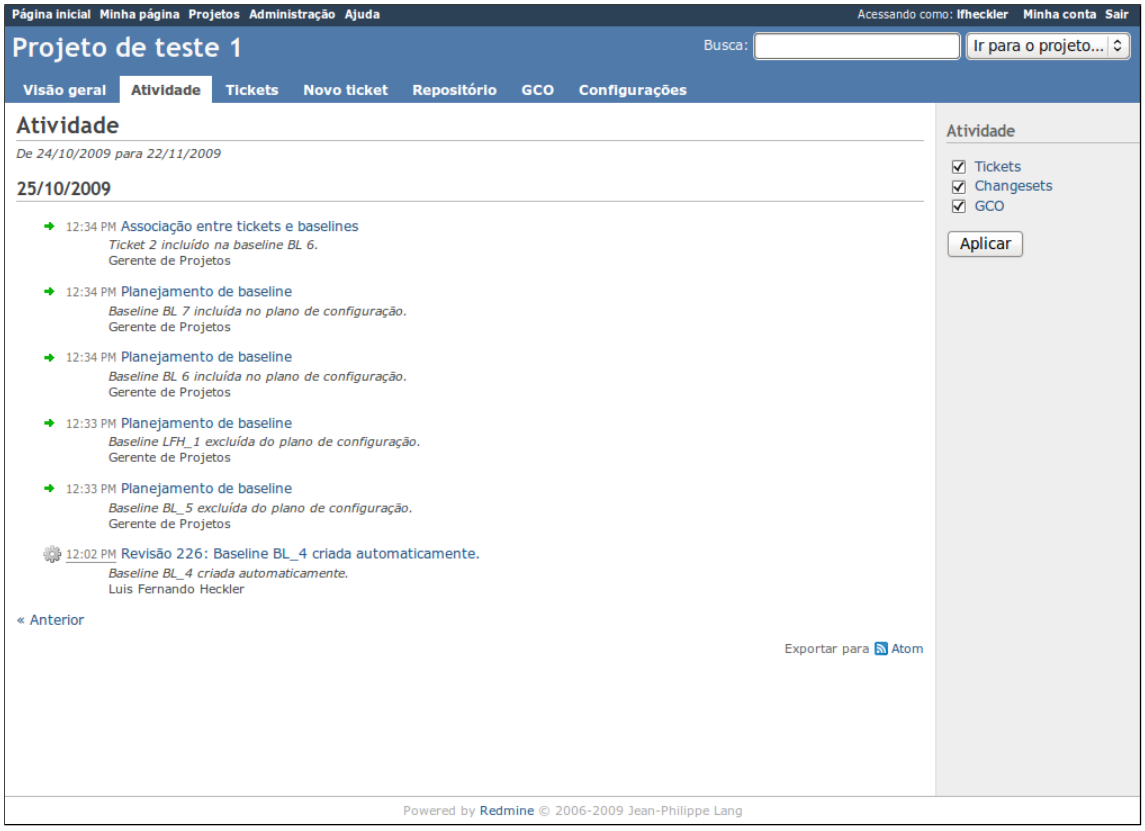

Figura 4.13: *Redmine GCO*, versão 1.0 - histórico de atividades realizadas

Fonte: elaborado pelo autor

Por fim, utilizando-se as funcionalidades do *Redmine* de controle de acesso configurável por perfil de uso do sistema, implementou-se o controle de segurança sobre os módulos desenvolvidos. As categorias de acesso disponibilizadas foram:

- Gerenciar configuração: permite que o usuário edite o plano de GCO, adicionando ou excluindo *baselines*, e associando revisões e *tickets*;

- Editar configuração: permite que o usuário acesse as aplicações da aba de GCO;
- Consultar configuração: permite que o usuário consulte mas não altere nenhum aspecto do planejamento de GCO ou das associações dos *tickets*;
- Criar *baseline*: permite que o usuário crie novas *baselines* previamente planejadas.

Sugere-se como abordagem inicial que o Gerente de Projetos receba as permissões de gerenciar e editar a configuração; que o Integrador do projeto possa criar *baselines*; e que o Auditor de Configuração possa consultar a configuração do projeto.

A Figura 4.14 a seguir demonstra o acesso restrito à seção de associação de *tickets* à baselines.

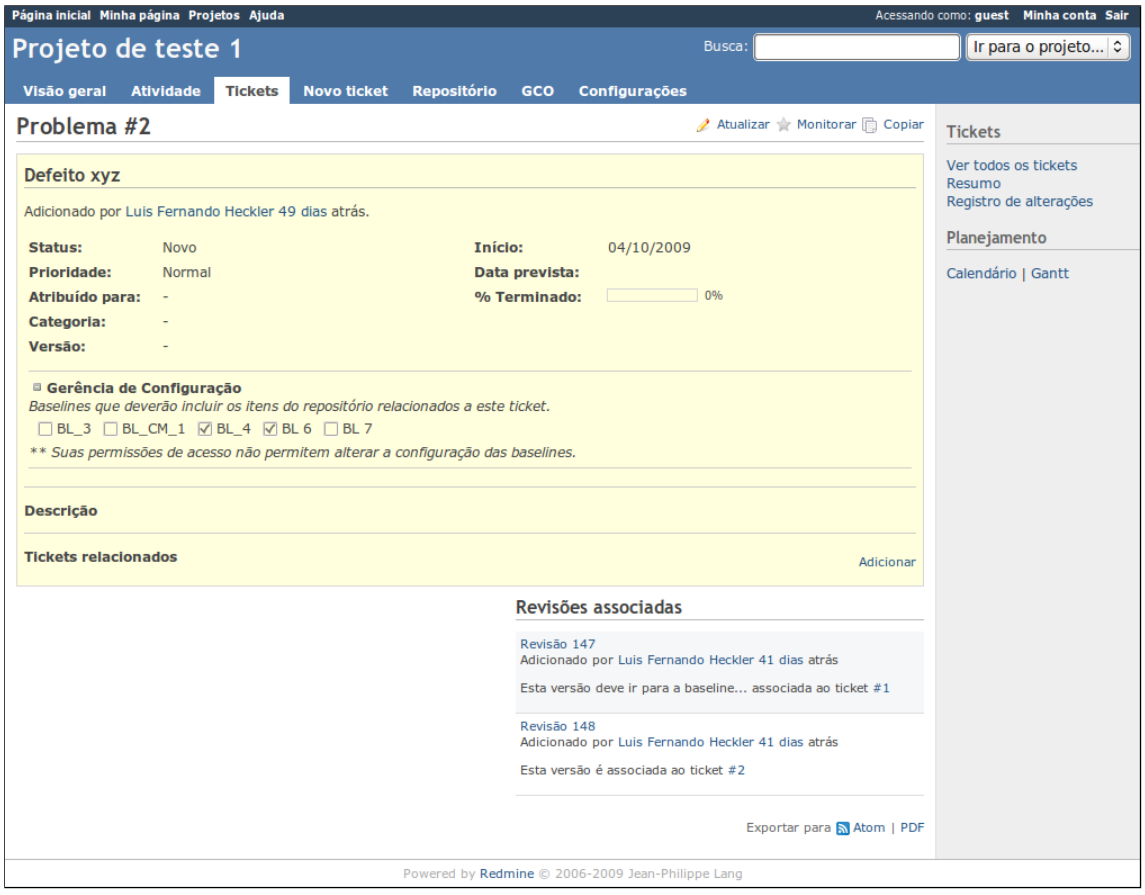

Figura 4.14: *Redmine GCO*, versão 1.0 - restrição de acesso

Fonte: elaborado pelo autor

Apresentadas estas características, considera-se que esta versão 1.0 atende os objetivos estabelecidos para este trabalho. No próximo capítulo são discutidas algumas considerações finais, bem como cita-se oportunidades de melhoria e perspectivas futuras vislumbradas a partir dos resultados desta experiência.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresenta-se como uma alternativa para complementar a ferramenta de gerência de projetos *Redmine* com novas funcionalidades voltadas para a área de Gerência de Configuração (GCO). Verificou-se a existência de tal demanda no contexto de implantação do modelo de qualidade de processo MPS.BR na empresa de desenvolvimento iProcess Soluções em Tecnologia. A pesquisa por soluções que atendessem os requisitos elicitados (descritos na seção 3.2) demonstrou a viabilidade de implementar-se uma solução própria, descrita neste trabalho.

Além disso, estima-se que muitas outras empresas de desenvolvimento de *software*, de pequeno e médio porte, tenham dificuldades semelhantes para implementar melhorias em seus processos internos, justamente pela carência de ferramentas a custos acessíveis que viabilizem a centralização das ações e informações. Desta forma espera-se que a solução desenvolvida, o *plugin Redmine GCO*, possa agregar valor e auxiliar estas empresas na melhoria da qualidade de seus processos e, portanto, de seus produtos.

A experiência do autor como projetista e desenvolvedor de sistemas na empresa alvo deste estudo demonstra que a aderência a processos de desenvolvimento como o MPS.BR traz de fato um ganho de qualidade nos produtos desenvolvidos. Mas é importante contar com sistemas e ferramentas que tornem este processo o mais natural possível no decorrer do trabalho de desenvolvimento. Percebeu-se este ganho na empresa a partir do aumento da centralização de funcionalidades na ferramenta *Redmine*, e estima-se que será ainda maior com a disponibilização deste novo módulo de GCO desenvolvido.

Infelizmente, por conta de demandas e priorizações internas, a empresa não pôde envolver-se em testes mais aprofundados de validação desta solução até o presente momento. Mas há a sinalização da gerência de utilizar o novo *plugin* em um próximo projeto piloto. Os resultados deste teste certamente serão utilizados para implementar novas melhorias na ferramenta, uma vez que o uso em situações de trabalho reais geralmente evidencia aspectos não percebidos durante as etapas de desenvolvimento, testes e homologação.

A seção a seguir apresenta algumas sugestões de melhorias para a ferramenta que foram identificadas no decorrer da elaboração deste trabalho. Estas deverão ser priorizadas posteriormente e, se aplicáveis, desenvolvidas futuramente.

5.1 Sugestões de melhorias

Nenhum sistema desenvolvido é tão perfeito que não possa ser melhorado, mesmo imediatamente após sua construção. A ferramenta desenvolvida no contexto deste

trabalho não é exceção. Algumas possíveis melhorias são decorrentes de decisões de projeto tomadas, visando validar conceitos primeiramente em uma versão mais simples, outras surgiram durante as etapas de testes e homologação, com o uso mais intenso do sistema.

A seguir lista-se alguns pontos que foram identificados pelo autor como oportunidades de melhoria da ferramenta *Redmine GCO*.

Refatoração e internacionalização

Algoritmos, quando escritos pela primeira vez, podem solucionar um determinado problema, mas se reavaliados certamente podem ser melhorados. No contexto do desenvolvimento desta solução certamente há pontos do código-fonte que podem ser melhorados visando um maior reuso, melhoria de desempenho ou mesmo aumento da clareza do código.

Um ponto já identificado para refatoração é a internacionalização dos textos do programa exibidos na interface com o usuário. Implementar toda a solução somente na Língua Portuguesa foi uma decisão planejada, visando a simplificação desta primeira versão. Porém, versões futuras, buscando um público-alvo mais abrangente, devem prever pelo menos a disponibilização da interface também na Língua Inglesa.

Suporte a outros sistemas de controle de versões

É pré-requisito atual para o uso da ferramenta desenvolvida que o Sistema de Controle de Versões dos projetos seja o *Subversion*. Como o *Redmine* suporta outros sistemas de versionamento, é fundamental que o *plugin* desenvolvido seja atualizado para suportá-los da mesma forma.

Melhorias de interface

Os testes e homologação com um ambiente que replica o ambiente de produção da empresa iProcess ressaltaram pontos nas interfaces que podem ser melhorados, aumentando a usabilidade do sistema. Sugere-se a criação de filtros intermediários nos pontos onde ocorre uma maior demora na resposta da aplicação por conta do volume de dados envolvido. Por exemplo, o módulo de associação de *tickets* e revisões mostrou-se consideravelmente lento ao lidar-se com projetos cujo repositório possui um grande número de revisões criadas.

Ainda, sugere-se avaliar quais os tipos de pesquisas mais comuns que os usuários do sistema (especialmente Auditor de Configuração e Gerente de Projetos) farão sobre estas novas informações de Gerência de Configuração. A partir desta avaliação podem ser criados novos relatórios e as telas existentes podem receber filtros adicionais.

Outro ponto interessante é implementar a possibilidade de edição do identificador das *baselines* planejadas. Na versão atual é necessário excluir e incluir novamente para alterar o nome descrito.

Processo de instalação e configuração

A versão atual exige a configuração de dois campos personalizados do projeto com identificadores específicos, para armazenar a informação de caminho no repositório do ramo principal e do ramo onde são salvas as *baselines*. O *script* de migração de dados utilizado na instalação deste *plugin* pode ser alterado de forma a criar

automaticamente estes novos campos.

Alternativamente, pode-se estudar a possibilidade de criar uma aplicação própria para armazenar estas e outras configurações específicas da ferramenta.

Correções no repositório

A versão da ferramenta desenvolvida permite corrigir a relação entre *tickets* e revisões, porém estes ajustes são salvos apenas no contexto do *Redmine* e não são propagados para o repositório. Desta forma, a informação dos ajustes realizados pode ser perdida caso o repositório do projeto no *Redmine* seja recriado. Recomenda-se que seja estudada uma forma de tornar permanentes estes ajustes, refletindo-os no repositório, ou disponibilizar uma forma de, ao ser recriado o repositório, serem igualmente recriados estes ajustes.

Histórico de atividades

As entradas no histórico do projeto geradas pelas atividades de GCO realizadas podem incluir uma justificativa quando for selecionada uma *baseline* que não seja a imediatamente próxima no planejamento, e também citar quando um artefato é removido da lista de objetos a serem copiados na criação da *baseline*.

5.2 Perspectivas futuras

Independentemente das sugestões de melhorias colocadas na seção 5.1 anterior, vislumbra-se algumas perspectivas para a solução *Redmine GCO* do ponto de vista do projeto em si, seu licenciamento e expectativa de contribuição à sociedade.

Desta forma o autor compromete-se que a ferramenta desenvolvida será disponibilizada como *software livre* assim que avaliar sob qual licença (GPLv2, GPLv3, LGPL, etc.) e em qual repositório de código (*SourceForge*, *Google Code*, etc.) será armazenada. Espera-se desta forma contribuir socialmente disponibilizando a solução gratuitamente, ao mesmo tempo que se abrirá espaço para outros desenvolvedores implementarem novas funcionalidades e desenvolverem a ferramenta ainda mais. O anúncio da disponibilização deverá ser feito em listas de discussão relevantes e, possivelmente, em artigo a ser publicado futuramente relatando esta experiência desenvolvida.

REFERÊNCIAS

BARTIE, A. **Garantia da qualidade de software**: adquirindo maturidade organizacional. 1ª.ed. Rio de Janeiro, Brasil: Elsevier, 2002.

BROOKS, F. P. J. **The mythical man-month - essays on Software Engineering - Anniversary Edition**. Boston, EUA: Addison Wesley, 2004.

BUCKLEY, F. J. **Implementing Configuration Management**: hardware, software and firmware. 2nd Ed..ed. [S.l.]: IEEE Press, 1996.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. **CMMI. Guidelines for Process Integration and Product Improve**. EUA: Addison Wesley, 2003.

CROSBY, P. B. **Quality is free**. 1th.ed. USA: McGraw Hill, 1979.

GUERRA, A. C.; COLOMBO, R. M. T. **Qualidade de produto de software**. 1ª.ed. Brasília, Brasil: Ministério da Ciência e Tecnologia, 2009.

HOUAISS, A.; VILLAR, M. d. S. **Minidicionário Houaiss da Língua Portuguesa**. 2ª.ed. Rio de Janeiro, Brasil: [s.n.], 2004.

IEEE. **Standard Glossary of Software Engineering Terminology**. [S.l.]: Institute of Electrical and Electronics Engineers, 1990.

IEEE. IEEE Standard for Software Configuration Management Plans. **IEEE Std 828-2005 (Revision of IEEE Std 828-1998)**, [S.l.], p.19 p., 2005.

KAN, S. H. **Metrics and Models in Software Quality Engineering**. 2nd.ed. Boston, EUA: Addison Wesley, 2002.

MAGALHãES, A. L. C. d. C. A Importância do Controle da Qualidade na Melhoria de Processos de Software. In: 2008, Campinas, Brasil. **Anais...** SOFTEX, 2008.

PAULA FILHO, W. d. P. **Engenharia de Software**: fundamentos, métodos e padrões. 2ª ed..ed. Rio de Janeiro, Brasil: LTC Editora, 2003.

PRESSMAN, R. S. **Software Engineering - A practitioner's approach**. 5th.ed. USA: McGraw Hill, 2001.

SAMARANI, P. R. d. M. **Um modelo de implementação do capability maturity model integration nível 2**. 2005. Dissertação — Universidade Federal do Rio Grande do Sul - UFRGS.

SOFTEX. **MPS.BR Melhoria de Processo do Software Brasileiro - Guia Geral versão MR-MPS:2009**. [S.l.]: SOFTEX, 2009.

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro - Guia de Implementação – Parte 2**: fundamentação para implementação do nível F do MR-MPS. 2009.

SOFTEX. **MPS.BR - Melhoria de Processos do Software Brasileiro**. Online; acessado em 02/11/2009, http://www.softex.br/mpsbr/_faq/faqDiversos.asp.

THOMAS, D.; HANSSON, D. H. **Agile Web Development with Rails**. 2nd Ed..ed. USA: The Pragmatic Programmers LLC, 2007.

APÊNDICE A PRÉ-REQUISITOS E INSTRUÇÕES PARA INSTALAÇÃO

Para a instalação do *plugin Redmine GCO* desenvolvido neste trabalho tem-se alguns pré-requisitos de componentes de terceiros e suas respectivas versões, a saber:

- *framework Rails* - versão 2.1.2[1]
- banco de dados *MySql* - versão 5[2]
- *Redmine* - versão 0.8[3]
- cliente *Subversion* - versão 1.6.6[4]
- biblioteca *systemu* [5]

Para a instalação do plugin desenvolvido deve-se seguir os seguintes passos:

1. Proceder com a instalação do *Redmine* e demais componentes de terceiros, cujos respectivos passos de instalação e configuração são divulgados por seus próprios responsáveis;
2. Copiar o código-fonte do *plugin Redmine GCO*[6] para a pasta "*vendor/plugins*" do *Redmine*;
3. Migrar o repositório de dados, executando-se o comando "*rake db:migrate_plugins RAILS_ENV=production –trace*" (no exemplo para o ambiente de produção);
4. Reiniciar o servidor HTTP do *Redmine*;
5. Acessar o *Redmine* com o usuário administrador e criar dois novos campos personalizados de projeto:
   - "Caminho para baselines"
   - "Ramo principal"
6. Configurar as permissões de acesso do novo plugin para os papéis no *Redmine*.

[1] Ver http://rubyonrails.org/
[2] Ver http://www.mysql.com/
[3] Ver http://www.redmine.org/
[4] Ver http://subversion.tigris.org/
[5] Ver http://rubyforge.org/projects/codeforpeople/. Pode ser instalada utilizando-se *ruby gem*.
[6] A ser disponibilizado como *software livre* em breve.