Recebido em: 15/03/09
Aprovado em: 07/05/10

Avaliação de uma ferramenta de controle de versões de *software* com o modelo de avaliação OPENBRR

Angela Maria Alves (CTI) - angela.alves@cti.gov.br
• Universidade Fedederal de Lavras – Caixa Postal 3037, CEP 37200-000, Lavras-MG, fone: (55) 35-3829-1122 / 1502
Ciniro Aparecido Leite Nametala (UFLA) - ciniro@gmail.com
Madison Delano Tobal da Paz (UFLA) - madison.paz@gmail.com
Maria Henrica Ruiter Kanamori (UFLA) - lukanamori@yahoo.com
Rodrigo Cardoso Mesquita (UFLA) - rcardosom@hotmail.com

Resumo
Este artigo tem como objetivo apresentar um caso de utilização de um modelo aberto que padroniza o processo de avaliação de produtos de software livre / código aberto. Para expor as características do modelo utilizou-se um caso de avaliação dos pontos fortes e fracos de um sistema de controle de versões de grande utilização no meio corporativo de desenvolvimento de software: o Subversion. O texto apresenta também as características do processo Configuration Management do modelo CMMI V1.1 que trata das boas práticas relacionadas a gerencia de configuração.
Palavras-chave: Avaliação; Software Livre; OPENBRR; Controle de Versão

Abstract
The purpose of this study is to present a case of using an open model that standardizes the process of evaluation of free open source software products . The model was used for the assessment of the strengths and weaknesses for a version control system used for software development: the Subversion. O text also presents the characteristics of the configuration management process from the CMMI V1.1 model.
Keywords: Assessment, open source, OPENBRR, configuration management.

GEPROS. Gestão da Produção, Operações e Sistemas – Ano 5, nº 3, Jul-Set/2010, p. 149-165

1. INTRODUÇÃO

Gerenciar uma equipe de desenvolvedores de *software* é uma tarefa que demanda um esforço incomensurável por parte das lideranças que se responsabilizam pela organização dos trabalhos a serem desempenhados. Os desafios começam quando se tenta conciliar, de acordo com as peculiaridades de cada indivíduo, os vários pensamentos, as culturas e as distintas formas de realizar uma mesma atividade. O problema se agrava na medida em que cada vez mais pessoas dessa equipe de desenvolvimento necessitam trabalhar, simultaneamente, em um mesmo projeto, dentro de um mesmo módulo, e até em um mesmo arquivo de código fonte a ser editado.

As alterações devem ser controladas para que o trabalho de um desenvolvedor não seja prejudicado pelas modificações indevidas introduzidas no código compartilhado, por parte de outro profissional também envolvido no projeto. Para evitar tal problema, foram desenvolvidos sistemas que, além de gerenciar o controle de mudanças no código-fonte, permitem que as versões utilizáveis desse *software* possam ser disponibilizadas e estendidas para futuras mudanças ou incrementos. Esses sistemas são os chamados Sistemas de Controle de Versões, ou VCS, do inglês *Versions Control Systems*.

Dentre os diversos Sistemas de Controle de Versões disponíveis, há opções comerciais, como o *SourceSafe*, da Microsoft, e o *ClearCase*, da IBM. Há também opções de *software* livre, como o *Concurrent Version Systems*, o CVS, e o seu sucessor, o *Subversion*, também conhecido por SVN. Esse último, bastante utilizado por desenvolvedores nas comunidades de *software* livre atuais, foi lançado em 2004, incorporando além das tradicionais características de concorrência do seu predecessor, o CVS, inovações que o tornram bastante eficiente no dia a dia dos desenvolvedores, como um tratamento mais eficiente de arquivos binários, mudanças no protocolo que permitem o envio bidirecional das diferenças nos arquivos, atomicidade nas operações de *commit*, entre outras.

Escolher uma dessas alternativas para implantar o controle de versões exige que sejam ponderados os pontos positivos e negativos de cada opção para que a escolha atenda às expectativas da equipe de desenvolvimento. Essa análise não pode ser feita à revelia ou subjetivamente, de acordo com as preferências individuais de quem está analisando. Daí a necessidade de criação de um modelo de avaliação aberto e padronizado que seja amplamente utilizado, sem controvérsias, possibilitando aos usuários do *software* livre compartilhar os resultados das avaliações. Um modelo padronizado permite um entendimento unificado dos resultados conseguidos nas avaliações. Sendo ele aberto, promove a confiança no processo de avaliação, assegurando sua flexibilidade para mudanças vindouras.

Nesse contexto, o *SpikeSource*, o Centro para Investigação de *Software* Livre na Carniege Mellon West e a Intel Corporation desenvolveram um modelo de levantamento para a avaliação de preparo para negócios, do inglês *Open Business Readness Rating*, ou OpenBRR, que permite aos profissionais da área de gestão tomar as decisões concernentes ao melhor aplicativo de *software* livre a ser utilizado em determinado projeto (BRR, 2005). Também permite que os usuários enviem suas avaliações para a comunidade de *software* livre, realimentando o repositório de informações, num crescente aprendizado possibilitado por esse *feedback*.

É esse modelo que será utilizado na avaliação do *Subversion*, comprovando ou não sua eficiência no controle de versão dos *softwares* livres desenvolvidos, de acordo com as pontuações consideradas pelo referido modelo OpenBRR.

O presente artigo é dividido em seis seções em sua totalidade. A primeira é essa introdução ao tema. A segunda seção aborda as características inerentes ao desenvolvimento distribuído e a importância do controle de versões para o sucesso nos projetos que envolvam compartilhamento de código. Na terceira seção a ferramenta *Subversion* é apresentada, explicitando-se as razões de sua adoção. A quarta seção expõe a fundamentação teórica do modelo de avaliação OpenBRR, para na quinta seção ser aplicada ao SVN, sendo o mote desse trabalho. Finalmente, na última seção, será apresentada a conclusão deste artigo, conforme a apreciação dos autores acerca das constatações levantadas na seção anterior.

2. DESENVOLVIMENTO DISTRIBUÍDO E CONTROLE DE VERSÕES

No processo *Configuration Management* (CM) do modelo CMMI V1.1 (CMMI, 2002) encontramos as boas práticas relacionadas com a gerencia de configuração.

O objetivo da gerência de configuração, área de processo de suporte, é estabelecer e manter a integridade dos produtos de trabalho usando a identificação de configuração, controle de configuração, contagem de *status* de configuração, e auditorias de configuração. Esta gestão permite que o desenvolvimento seja feito por recursos distribuídos nos diversos continentes, facilitando, assim, montar a equipe de desenvolvedores com os melhores especialistas.

Nesta área de processo têm-se três (3) metas específicas (também conhecidas por *Specific Gold* (SG)) e para cada meta teremos as praticas específicas (também conhecidas por *Specific Pratices* (SP)) e finalmente têm-se as sub-práticas respectivamente (também conhecidas por *Subpractices*).

A seguir serão relacionados por meta específica, as práticas especificas e as sub-práticas respectivamente.

SG1: Estabeleça *baselines* onde os *baselines* dos produtos de trabalho identificados são estabelecidos.

SP-1.1: Identifique os itens de configuração.

1. Selecione os itens de configuração e os produtos de trabalho que os compõem baseado em um critério documentado.
2. Defina identificadores únicos para os itens de configuração.
3. Especifique as características importantes de cada item de configuração (autor, tipo, linguagem de programação).
4. Especifique quando cada item de configuração é colocado sob gerenciamento de configuração.
5. Identifique a pessoa responsável (*owner*) por cada item de configuração.

SP-1.2: Estabeleça um sistema de gerenciamento de configurações.

1. Estabeleça um mecanismo para gerenciar os múltiplos níveis de controle do gerenciamento de configurações.
2. Armazene e recupere os itens de configuração do sistema de gerenciamento de configurações.
3. Compartilhe e transfira itens de configuração entre níveis de controle dentro do sistema de gerenciamento de configurações.
4. Armazene e recupere versões arquivadas de itens de configuração.
5. Armazene, atualize e recupere os registros de gerenciamento de configuração.
6. Crie relatórios de gerenciamento de configurações extraídos do sistema de gerenciamento de configurações.
7. Preserve o conteúdo do sistema de gerenciamento de configurações.
8. Revise a estrutura de gerenciamento de configurações conforme necessário.

SP-1.3: Crie ou libere os baselines.

1. Obtenha autorização do CCB (Comitê de Controle de Configurações) antes de criar e liberar *baselines* de itens de configuração.
2. Crie ou libere *baselines* apenas para itens de configuração existentes no sistema de CM.

SG2: Acompanhe e controle mudanças quando houver modificações nos produtos de trabalho sob a gerência de configurações.

SP-2.1: Acompanhe as requisições de mudanças.

1. Inicie e grave requisições de mudanças na base de dados de requisições de mudanças.
2. Analise o impacto das mudanças e consertos propostos nas requisições de mudanças.
3. Revise as requisições de mudanças que serão endereçadas no próximo *baseline*.
4. Acompanhe o *status* das requisições de mudanças até o fechamento.

SP-2.2: Controle os Itens de Configuração.

1. Controle as mudanças nos itens de configuração durante a vida do produto.
2. Obtenha a autorização apropriada antes de inserir um item de configuração alterado no Sistema de CM.
3. *Check in* e *Check out* dos itens de Configuração do Sistema de CM para incorporar as mudanças de maneira que se mantenha a correção e integridade dos itens de configuração.
4. Realize revisões para garantir que as mudanças não causaram efeitos indesejáveis nos *baselines*.
5. Registre as mudanças dos itens de configurações e as razões para as mudanças conforme apropriado.

SG3: Estabeleça Integridade onde a integridade dos *baselines* é estabelecida e mantida.

SP-3.1: Estabeleça registros de Gerenciamento de Configuração.

1. Registre as ações de CM em detalhes suficientes para que o conteúdo e *status* de cada item de configuração são conhecidos e versões anteriores possam ser recuperadas.
2. Garanta que os relevantes *stakeholders* tenham acesso e conhecimento sobre o *status* dos itens de configuração.
3. Especifique a última versão dos *baselines*.
4. Identifique as versões dos Itens de Configuração que constituem um *baselines* particular.
5. Descreva as diferenças entre *baselines* sucessivos.
6. Revise o *status* e a história de cada item de configuração conforme necessário.

SP-3.2: Realize Auditorias de Configuração.

1. Avalie a integridade dos *baselines*.
2. Confirme que os registros de configuração identificam corretamente as configurações dos Itens de configuração.
3. Revise a estrutura e a integridade dos itens no sistema de CM.
4. Confirme se os Itens no sistema de CM estão completos e corretos.
5. Confirme a conformidade com os padrões e procedimentos de CM aplicáveis.
6. Acompanhe itens de ação oriundos de auditorias até o fechamento.

3. A FERRAMENTA *SUBVERSION*

Anteriormente ao *Subversion* (SVN), o *Concurrent Version System* (CVS) foi amplamente utilizado em todo o mundo. Sua premissa era ser um *software* livre para suportar o desenvolvimento em grandes redes distribuídas, o que o tornou muito popular.

Muitos usuários encontraram limitações no CVS, pois ele não possui um controle adequado em operações como mover ou renomear arquivos e controle de diretórios; tarefa complementar, dentre muitas outras, que o SVN possui suporte nativo. O SVN foi desenvolvido para ser o sucessor do CVS. Robusto e com o desenvolvimento totalmente planejado pela CollabNet Inc[1] e por especialistas no CVS, foi construído para funcionar de forma semelhante ao CVS para que os antigos usuários migrassem ao novo sistema com relativa facilidade. Nasceu então uma ferramenta que cada vez mais é utilizada tanto no mundo do *software* livre quanto em ambientes corporativos: o *Subversion*.

1 Empresa que desenvolve e comercializa software para a colaboração distribuída a nível mundial entre as organizações de desenvolvimento de software. Seu site oficial é o http://www.collab.net.

3.1. A História do *Subversion*

No início de 2000, a CollabNet, Inc. começou a procurar desenvolvedores para escrever um substituto para o CVS. CollabNet oferece um pacote de *software* chamado *Source Cast*, no qual um componente é de controle de versão. Embora o *Source Cast* usasse o CVS como seu controle de versão inicial, as limitações do CVS foram óbvias no começo, e a CollabNet percebeu que teria de encontrar algum produto com mais recursos. Mesmo assim, o CVS acabou tornando-se um padrão no mundo *open source* porque não existia nada superior, ao menos não em uma licença livre. Por causa disso, a CollabNet decidiu escrever um novo sistema de controle de versão do zero, aproveitando as ideias do CVS, mas sem os *bugs* e falta de recursos.

Em fevereiro de 2000, eles contataram Karl Fogel, o autor de Desenvolvimento *Open Source* com CVS (FORGEL, 2007), e perguntaram-lhe se gostaria de trabalhar neste novo projeto. Coincidentemente nessa época, Karl já estava discutindo um projeto para um novo sistema de controle de versão com seu amigo Jim Blandy. Em 1995, os dois começaram a *Cyclic Software*, uma empresa que provia contratos de suporte a CVS, e embora mais tarde eles tenham vendido o negócio, eles ainda utilizavam o CVS todos os dias em seus empregos. Sua frustração com o CVS levou Jim a pensar com cuidado em melhores maneiras de gerenciar versões, e ele não só criou o nome "*Subversion*", mas também o projeto básico do repositório de arquivos. Quando a CollabNet entrou em contato, Karl imediatamente concordou em trabalhar no projeto, e Jim tornou-se seu auxiliar. A CollabNet contratou Karl e Ben Collins-Sussman, e detalhou o projeto iniciado em maio. Com a ajuda de *know-how* de alguns bons produtos de Brian Behlendorf e Jason Robbins da CollabNet, e Greg Stein (desenvolvedor independente ativo no processo de especificação do WebDAV/DeltaV), o *Subversion* rapidamente atraiu a comunidade de desenvolvedores. Veio a tona que muitas pessoas tiveram a mesma frustração com o CVS, e deram boas-vindas à chance de finalmente fazerem algo quanto a isso.

A equipe do projeto original focou em algumas metas simples. Eles não queriam desenvolver uma nova metodologia de controle de versão, eles queriam apenas consertar o CVS. Decidiram que o *Subversion* deveria ter os mesmos recursos que o CVS, e preservar o mesmo modelo de desenvolvimento, mas não duplicar as falhas óbvias do sistema anterior. E mesmo não sendo necessário ser um substituto, o novo sistema deveria ser similar para que qualquer usuário CVS pudesse fazer a migração com pouco esforço.

Depois de quatorze meses de desenvolvimento, o *Subversion* se tornou "auto-hospedável" em 31 de agosto de 2001. Isto quer dizer que os desenvolvedores pararam de utilizar o CVS para gerenciar o código do sistema em desenvolvimento e passaram a utilizar o próprio *Subversion*.

A licença de *copyright* da CollabNet é totalmente compatível com a linhas de *software* livre *Debian*. Em outras palavras, todos podem modificar, baixar e redistribuir o *Subversion*; não é preciso permissão da CollabNet ou de ninguém para isso.

3.2. Definição do *Subversion*

O *Subversion* é um sistema de controle de versão livre desenvolvido pela CollabNet Inc. em 2001, com vistas a substituir o CVS. Isso quer dizer que ele gerencia arquivos e diretórios no decorrer do tempo. Possui um repositório semelhante a um servidor de arquivos comum, exceto pelo fato de lembrar cada mudança feita nos arquivos e diretórios. Isto permite recuperar antigas versões dos dados ou examinar a história de mudanças de qualquer alteração. Como analogia, muitos entendem o controle de versão como uma espécie de "máquina do tempo", em que se torna possível voltar a informação a praticamente qualquer ponto de mudança do passado.

É possível acessar o repositório através de redes drasticamente distribuídas, o que permite que seja utilizado por pessoas em diferentes computadores com um desempenho muito bom. Há suporte para pessoas modificarem e gerenciarem o mesmo conjunto de dados de modo colaborativo, sendo que o SVN consegue gerenciar além das alterações, os conflitos eventuais de edição que podem ocorrer quando mais de um usuário altera o mesmo conteúdo do repositório. O trabalho desenvolve-se mais rápido pelo simples fato do controle de modificações estar disponível. E por o trabalho estar com versões enumeradas, não há temor quanto ao prejuízo de qualidade por perda de conteúdo – se algo está incorreto é só retornar a um ponto anterior.

Alguns controles de versões são também sistemas de gerenciamento de configuração (SCM). Estes sistemas são especificamente desenvolvidos para gerenciar árvores de código fonte, e têm muitos recursos específicos para o desenvolvimento de *software* – tais como a compreensão nativa de linguagens de programação, ou a disponibilização de ferramentas para compilar o *software*. O *Subversion*, por muito tempo, não foi um desses sistemas. Ele foi construído inicialmente como um sistema geral que pode ser usado para gerenciar qualquer coleção de arquivos. Para você, aqueles arquivos podem ser código-fonte – para outros, alguma coisa da lista de itens da loja ou uma lista de *downloads*.

Recentemente, a CollabNet lançou a versão 1.5 do *Subversion*, integrada a um conjunto de outros *softwares*, com recursos de um SCM completo.

Claramente o SVN é uma alternativa excelente para desenvolvedores que querem gerenciar código fonte, pois além de gerenciar os arquivos do repositório e ter um controle de concorrência superior a muitos sistemas comerciais, possui custo zero.

Assim, de acordo com os argumentos acima, e com a escolha do *software* definida, a próxima etapa é conhecer o modelo de avaliação que será utilizado para comprovar sua eficiência na prática. A próxima seção aborda o modelo adotado neste artigo.

GEPROS. Gestão da Produção, Operações e Sistemas – Ano 5, nº 3, Jul-Set/2010, p. 149-165

4. MODELO DE AVALIAÇÃO OPENBRR

Criado pela Universidade Carnegie Mellon, o modelo OpenBRR, do inglês *Open Business Readiness Rating*, é aberto e segue um padrão avaliativo que visa avaliar qualquer tipo de sistema de *software* de maneira formal, além de agilizar os processos de adoção de *software* livre e propor um modelo conceitual para análise com base nos propósitos do cliente. Este modelo tem tido uma crescente adoção devido às características de padronização que ele oferece. Isso se deve ao fato de que o modelo possui uma linguagem comum que traz nivelamento de entendimento nos resultados gerados. O modelo OpenBRR agrega várias características benéficas e, por ser aberto, auxilia na expansão, mudanças e melhorias na avaliações possivelmente feitas pelos usuários, resultando em grande confiança no processo de avaliação devido à transparência das métricas empregadas (BRR, 2005).

Este modelo possibilita que os riscos de utilização de um *software* possam ser diminuídos quando na adoção do mesmo, visto que o OpenBRR permite que a avaliação de um sistema pode ser compartilhada e padronizada dentro de uma mesma linguagem. Segundo Moraes (2008), o OpenBRR se torna atraente, também, para os desenvolvedores de *Software* Livre em relação à melhoria dos pontos fracos identificados através da avaliação.

Baseado em quatro etapas o modelo OpenBRR permite que através de cada fase o *software* seja avaliado passo-a-passo. Sendo estas:

- A primeira fase consiste em uma avaliação para escolher o grupo de pacotes de *software* a ser avaliado, dentre alguns candidatos selecionados previamente, atendendo aos requisitos do cliente.
- Na segunda fase do modelo é feita uma organização das métricas e categorias que serão utilizadas na avaliação na medida em que prioridades são determinadas.
- A terceira fase é caracterizada pela coleta e processamento dos dados observados.
- A quarta fase é a adaptação dos dados para o OpenBRR (BRR, 2005).

A figura 1 abaixo ilustra o modelo OpenBRR e suas quatro etapas.

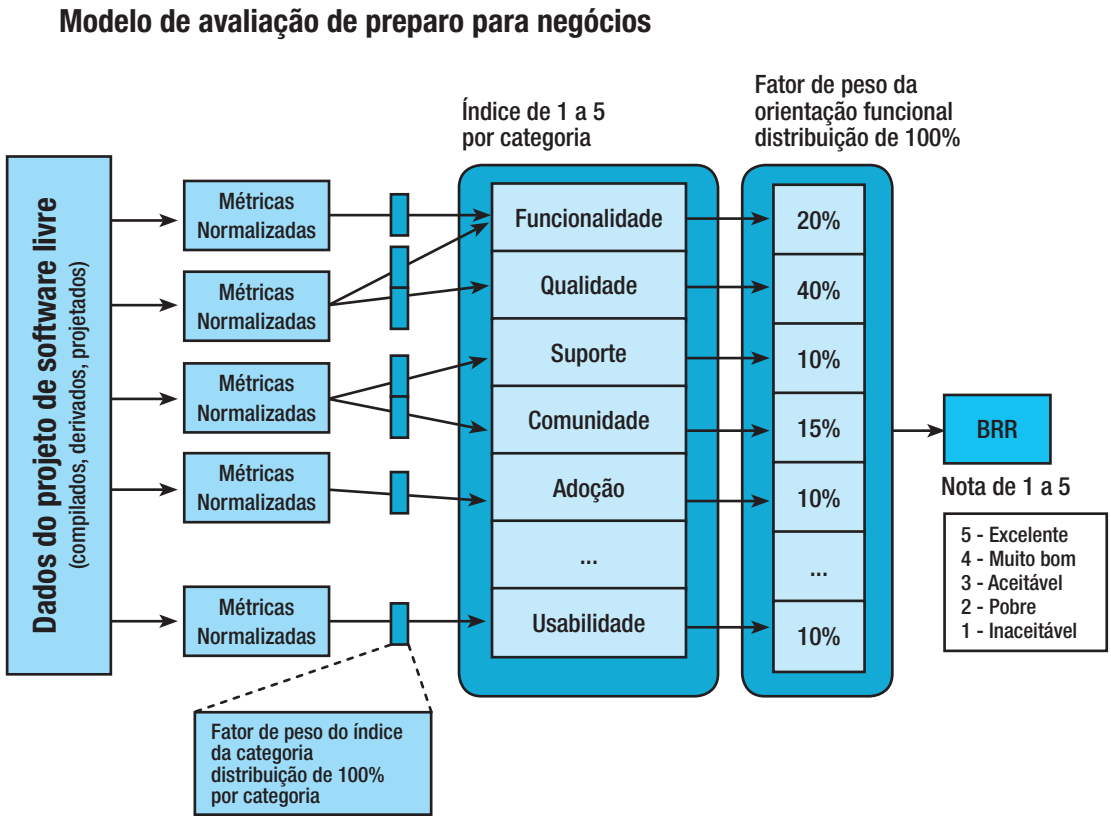

Figura 1 – Etapas do Modelo de Avaliação.
Fonte: OpenBRR (BRR, 2005)

A próxima seção aprofunda o modelo aplicando seus conceitos na avaliação do *software* de controle de versão escolhido, o *Subversion*.

GEPROS. Gestão da Produção, Operações e Sistemas – Ano 5, nº 3, Jul-Set/2010, p. 149-165

5. AVALIAÇÃO DO *SUBVERSION* USANDO O OPENBRR

Adaptando o modelo OpenBRR às necessidades de avaliação proposta neste artigo, não foi utilizada a primeira fase do modelo, definida como "avaliação rápida" dos pacotes de *software* com vistas à escolha dos *softwares* que serão analisados. Isso se deve ao fato de que o *software* a ser avaliado já foi escolhido, conforme os argumentos expostos nas seções anteriores.

Iniciando a segunda fase, intitulada "definição de métricas e categorias", propõe-se a utilização de métricas que correspondam às propriedades mensuráveis, quantitativas e qualitativas, do *software*. É importante considerar que as métricas qualitativas deverão ser normalizadas para que produzam dados mensuráveis, conforme Moraes (2008).

Baseando-se no trabalho de Moraes (2008), foram utilizadas doze categorias de levantamento, mostradas na tabela 1. O documento OpenBRR aconselha que sejam avaliadas no máximo sete categorias. Após a enumeração das categorias, o modelo orienta que seja atribuído um percentual de importância a cada categoria escolhida para análise totalizando 100% na soma entre elas (BRR, 2005). As categorias não utilizadas na avaliação estão marcadas com um sinal de asterísco (*).

Com o levantamento das categorias realizado, segue-se para a terceira fase do modelo OpenBRR, denominada de "coletas de dados e resultados", onde houve a deliberação das métricas, sugeridas pelo modelo, em cada categoria levantada e escolhida. As tabelas de 2 a 7 mostram, respectivamente, as categorias usabilidade, *performance*, segurança, qualidade, escalabilidade e documentação, e suas métricas relacionadas com os pesos definidos a elas e às suas sub-categorias.

A tabela é preenchida nos campos "*Escore* sem peso", atribuindo-se notas de 1 a 5, variando de acordo com a característica de cada métrica aplicada, através das considerações já definidas pelo modelo OpenBRR[2]. Conforme essas métricas sejam pontuadas, a totalização da categoria é ponderada em conformidade com os pesos previamente definidos para cada métrica, gerando o valor final por categoria, denominado "*Escore* com peso". As seguintes tabelas apresentam os resultados encontrados.

2 Os dados que serviram de base para a pontuação foram obtidos no próprio sítio do Subversion, na área que relata as mudanças advindas em cada versão: http://svn.collab.net/repos/svn/trunk/CHANGES

Tabela 1 – Categorias escolhidas e com os pesos atribuídos.

Tecnologia avaliada		
Nome do Componente: *Subversion*		
Tipo do Componente: Controle de Versões		
Modo de Uso: Uso regular		
Posição	Categoria	Peso
1	Funcionalidade	25,00%
2	Usabilidade	15,00%
3	Qualidade	15,00%
4	Segurança	15,00%
5	*Performance*	10,00%
6	Escalabilidade	10,00%
7	Arquitetura*	0,00%
8	Suporte*	0,00%
9	Documentação	10,00%
10	Adoção*	0,00%
11	Comunidade*	0,00%
12	Profissionalismo*	0,00%
	Peso Total	100,00%

Fonte: BRR (2005).

Tabela 2 – Categoria Usabilidade e suas métricas

Título da Categoria			Peso	*Escore* sem peso	*Escore* com peso
Usabilidade			15,00%	4,5	0,675
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Experiência do usuário final	Simples e Intuitivo	50,00%	5	2,5	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Tempo para configurar os pré-requisitos para instalar o pacote de *software* livre	20 minutos	25,00%	4	1	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Tempo para a instalação / configuração mais comum	10 minutos	25,00%	4	1	

Tabela 3 – Categoria *Performance* e suas métricas

Título da Categoria			Peso	*Escore* sem peso	*Escore* com peso
Performance			10,00%	4	0,4
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Teste de desempenho e relatórios de *Benchmark*	Sim	50,00%	3	1,5	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Ajuste de desempenho e configuração	Sim, algum	50,00%	5	2,5	

Tabela 4 – Categoria Segurança e suas métricas

Título da Categoria			Peso	*Escore* sem peso	*Escore* com peso
Segurança			15,00%	4,2	0,63
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de vulnerabilidades nos últimos 6 meses que são moderados ou extremamente críticas	0	60,00%	5	3	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de vulnerabilidades de segurança ainda abertas (sem *patches*)		0,00%	0	0	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Há informações (página na *web*, *wiki*, etc) dedicadas à segurança?	Sim	40,00%	3	1,2	

Tabela 5 – Categoria Qualidade e suas métricas

Título da Categoria			Peso	*Escore* sem peso	*Escore* com peso
Qualidade			15,00%	3,8	0,57
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de versões menores nos últimos 12 meses	4	25,00%	1	0,25	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de versões de ponto/*patches* nos últimos 12 meses	2	10,00%	3	0,3	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de *bugs* abertos nos últimos 6 meses	4	30,00%	5	1,5	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de *bugs* concertados nos últimos 6 meses (comparados com o número de *bugs* abertos)	90%	30,00%	5	1,5	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Número de *bugs* P1 / críticos abertos	0	5,00%	5	0,25	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Média de idade dos *bugs* P1 nos últimos 6 meses		0,00%	0	0	

Tabela 6 – Categoria Escalabilidade e suas métricas

Título da Categoria			Peso	*Escore* sem peso	*Escore* com peso
Escalabilidade			10,00%	5	0,5
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Distribuição de Referência	http://subversion.tigris.org/	50,00%	5	2,5	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Projetado para escalabilidade	Sim, extensivo	50,00%	5	2,5	

Tabela 7 – Categoria Documentação e suas métricas

Título da Categoria			Peso	*Escore* sem peso	*Escore* com peso
Documentação			10,00%	5	0,5
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Existência de várias documentações	Possui até livros *on-line*	70,00%	5	3,5	
Nome da Métrica	Dados obtidos	Peso	*Escore* sem peso	*Escore* com peso	
Framework para as contribuições dos usuários	Pessoas podem contribuir	30,00%	5	1,5	

Tabela 8 – Categoria Funcionalidade e suas métricas.

Tecnologia Avaliada			
Nome do Componente: *Subversion*			
Título da Categoria			Pontuação
Funcionalidade			5
Source			
http://subversion.tigris.org/			
Funcionalidades-padrão	Peso	Pontuação	
Gerenciamento do Processo de Configuração			
Guia para o processo de Configuração	3	2	
Planejamento para a Configuração	3	3	
Plano de Gerenciamento de Configuração	2	-2	
Identificação da Configuração do *Software*			
Identificando itens a serem controlados	3	3	
Item de configuração de *software*	3	3	
Versão de *software*	3	3	
Baseline	2	2	
Aquisição de itens de configuração	3	3	
Documentação	2	2	
Controle de configuração de *Software*			
Processo de requisição de mudança	2	2	
Implementação das mudanças	3	2	
Desvios e Renúncias	1	-1	
Contabilização do *Status* do *Software*			
Informação do *Status* de Configuração do *Software*	2	2	
Relatório de *Status* de Configuração do *Software*	1	1	
Funcionalidades Extra	Peso	Pontuação	
Versionamento de Diretórios	3	2	
Histórico de versões reais	2	2	
Commits atômicos	2	2	
Versionamento de Metadados	1	1	
Escolha de rede em camadas	2	2	
Manuseio consistente de dados	3	2	
Branching e *Tagging* eficientes	3	2	
Possibilidade de *hackeamento*	2	1	
	Peso Total	Pontuação Total	Percentual
	33	39	118,18%

Fonte: IEEE (2004); SUSSMAN (2004)

A categoria "funcionalidade" é distintamente pontuada pelo modelo OpenBRR, com relação às demais categorias. Conforme Moraes (2008), as funcionalidades esperadas no *Subversion* foram definidas baseadas no capítulo sete do IEEE (IEEE, 2004) no que tange a Gerência de Configuração e também pelas informações colhidas em Sussman (2004).

Utilizando o *template*[3] do modelo de avaliação, foram determinados os conjuntos de recursos esperados, nomeados pelo modelo de funcionalidades-padrão. Uma pontuação para cada uma das funcionalidades-padrão e extras, de 1 a 3 foram estabelecidas: o valor 1 significa irrelevante, o valor 2, pouco importante e o valor 3, muito importante. Caso a funcionalidade-padrão não fosse encontrada, o valor do índice negativo ao campo da nota da funcionalidade seria atribuído. Para cada funcionalidade extra encontrada no *Subversion* foi adicionada o índice de importância a ela e 0, caso não estivesse presente na ferramenta.

De acordo com (BRR, 2005), a nota final da categoria "funcionalidade" é composta da divisão da soma cumulativa das notas obtidas pelo total de pontos que poderiam ser obtidos somente com o conjunto de funcionalidades-padrão. O intervalo da nota que o *software* poderia assumir está entre porcentagens menores do que 0% ou maiores do que 100%. Este intervalo foi normalizado, em cinco notas (de 1 a 5) dependendo da faixa em que a nota esteja inserida, para compor a nota final da categoria. A tabela 8 mostra os resultados das atribuições de pontos na referida categoria.

Somando-se cada uma das pontuações obtidas nas sete categorias, chega-se a uma marca de 4,525, que é uma pontuação considerável, segundo o modelo BRR. Assim, chega-se efetivamente à etapa final do modelo OpenBRR, definida como "conclusões através da tradução dos dados", onde a pontuação pode ser utilizada para comparações com outras ferramentas também avaliadas por esse modelo aberto. Na próxima seção, evidenciam-se as reflexões e resultados da referida análise do *software* de controle de versões, o *Subversion*, concluindo o presente artigo.

6. CONCLUSÕES

O desenvolvimento distribuído de produtos de *software*, dada a sua complexidade crescente, exige dos gestores a utilização de métodos e ferramentas que auxiliem no processo de armazenamento, controle de modificações no código fonte compartilhado, nas versões dos sistemas disponibilizados e na gestão e distribuição de todos os documentos do produto e do projeto, permitindo não somente a maior eficiência nas atualizações subsequentes, como também evitando retrabalho ocasionado por alterações acidentais indevidas em códigos já funcionais existentes. Escolher uma alternativa de controle de versões dentre as diversas opções existentes é uma tarefa crucial para o sucesso do projeto. Para tanto, foi apresentado, no artigo, um modelo aberto e padronizado de avaliação de produtos de *software* livre ou de código aberto, o OpenBRR, aplicado na avaliação da viabilidade de utilização do *Subversion*, sistema de controle de versões substituto do CVS, em projetos que demandam a distribuição no desenvolvi-

3 O modelo de pontuação pode ser acessado em http://www.openbrr.org/wiki/index.php/Downloads

mento. De acordo com a avaliação feita no *software*, após meticulosa análise dos dados, fica confirmada a alta marca final obtida com a soma das categorias escolhidas, chegando-se a um valor de 4,525. Isso confirma o pensamento inicial dos autores do artigo de que o *Subversion* é uma ferramenta que provê ganhos de produtividade no gerenciamento de versões de *software* em desenvolvimento, permitindo um trabalho distribuído seguro e eficaz, ainda mais sendo uma plataforma livre.

Ressalta-se, ainda, que a avaliação realizada neste artigo não contempla o estudo de todos os requisitos possíveis do *Subversion*. Foram usadas as métricas sugeridas pelo modelo OpenBRR, porém elas podem ser redefinidas de acordo com o interesse de quem vai utilizar o *software*. Assim, determinados aspectos podem não ter sido observados de acordo com a necessidade em particular de outros avaliadores que pretenderão adotar o SVN.

Pelo exposto, a comprovada eficácia do *Subversion* nos critérios definidos na avaliação pelo modelo OpenBRR corrobora com o já aclamado sucesso desse sistema de controle de versões nas comunidades de *software* livre e nas corporações privadas. Porém, fica evidente a necessidade de manutenção de um repositório com informações mais completas e diversificadas, que atendam às diferentes necessidades de cada utilizador do modelo, segundo seus critérios específicos.

7. REFERÊNCIAS BIBLIOGRÁFICAS

BRR. **Business Readiness Rating Whitepaper**. Modelo de Levantamento para Avaliação de Preparo para Negócios: uma proposta de padrões abertos para facilitar a avaliação e adoção de soluções de software livre. BRR 2005 RFC-1. Tradução por Alexandre Rocha Lima e Marcondes. Disponível em: <http://www.openbrr.org/wiki/images/5/59/BRR_whitepaper_2005RFC1-pt-BR.pdf>. Acesso em 15 de julho de 2008.

FOGEL, K.; BARR, M. **Open Source Development with CVS 3rd edition**. Ed. Coriolis, 2007. USA. Disponível em: < http://cvsbook.red-bean.com/OSDevWithCVS_3E.pdf>. Acesso em 17 de julho de 2008.

IEEE. **Computer Society**. *Software* Engineering Body of Knowledge (SWEBOK).EUA: Angela Burgess, 2004. Disponível em: <http://www.swebok.org>. Acesso em 20 de julho de 2008.

MORAES, A. C. R. **Avaliação de Ferramentas Livres de Apoio à Gestão de Requisitos e Testes Utilizando o Modelo de Avaliação OpenBRR**. Departamento de Ciências da Computação, Universidade Federal de Lavras – UFLA. Lavras, MG, Brasil: (s.n.), 2008. Disponível em: <http://www.esl.nteufla.com/moodle/mod/resource/view.php?id=208>. Acesso em 16 de julho de 2008.

SEI. CMMI. **The Capability Maturity Model for Software**. Version 1.1- CMU/SEI-2002-TR-012, Março de 2002. Disponível em: < http://www.sei.cmu.edu >. Acesso em 23 de julho de 2008.

SUSSMAN, B. C.; FITZPATRICK, B. W.; PILATO, C. M. **Version Control with Subversion: for Subversion 1.4** (Compiled from R2866). Ed. O'Reilly, 2004, USA. Disponível em: <http://svnbook.red-bean.com/en/1.4/svn-book.pdf>. Acesso em 14 de julho de 2008.