Elaboração de processos de Gerência de Configuração de Software: Uma proposta de aplicação no ambiente iNtegra

Muilaerte Ferreira de Vasconcelos Júnior

JUIZ DE FORA
Dezembro, 2011

Elaboração de processos de Gerência de Configuração de Software: Uma proposta de aplicação no ambiente iNtegra

Muilaerte Ferreira de Vasconcelos Júnior

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharel em Ciência da Computação

Orientador: Evaldo de Oliveira da Silva

JUIZ DE FORA
Dezembro, 2011

Elaboração de processos de Gerência de Configuração de Software: Uma proposta de aplicação no ambiente iNtegra

Muilaerte Ferreira de Vasconcelos Júnior

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

_____

Prof. Evaldo de Oliveira da Silva, M. Sc.
Orientador

_____

Prof. Giuliano Prado de Morais Giglio, M. Sc.

_____

Daves Marcio Silva Martins, M. Sc.

JUIZ DE FORA, MG – BRASIL
Dezembro, 2011

AGRADECIMENTOS

Agradeço a todos os leitores que interromperam seu tempo para ler este agradecimento, isto demonstra o interesse na minha jornada que percorri até a produção deste trabalho.

A minha família, meu pai, minha mãe, minha irmã, meu irmão e a Milla por terem me apoiado durante todos esses anos com amor, educação e incentivo.

A minha namorada Thais por todo apoio, amor e carinho que tem me dado durante nossos anos de relacionamento.

Ao meu orientador Evaldo por ter aceitado o convite para orientação assim como toda paciência, ensinamentos proporcionado no período deste trabalho.

Aos meus amigos da UFJF em que citá-los poderia provocar o esquecimento de alguns nomes, pela amizade e experiência proporcionada durante o período do curso e que esta dure ao longo de nossas vidas.

Aos amigos do NRC/iNtegra: Daves, Tatiane, Rubens, Eduardo Barrére, Alcindo, Thiago e demais pelo pronto atendimento em ajudar, amizade e ensinamentos.

Aos demais amigos por todo incentivo e apoio nos momentos de descontrações.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A gerência de configuração de software, segundo modelos de maturidade, é uma área chave para elevar a qualidade de software, tendo como processos básicos o controle de mudanças e de versões. Para a implantação destes processos faz-se necessário o apoio de uma documentação que apresente propostas para implantação de acordo com as características de um ambiente qualquer. Este trabalho aborda os principais conceitos relacionados para a implantação de processos de gerência de configuração e apresenta uma proposta de implantação para os processos de controle de versões e de mudanças para o projeto iNtegra desenvolvido na Universidade Federal de Juiz de Fora no Instituto de Ciências Exatas. O processo de gestão de mudanças tem como principal objetivo controlar as solicitações de demandas e o processo de controle de versões almeja manter a integridade da *baseline*, apoiar a escolha de estratégia de ramificações e melhorar a eficiência do desenvolvimento.

Lista de Reduções

BSD	*Berkeley Software Distribution*
CI	*Configuration Item*
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integration*
CSV	*Comma-Separated Values*
ES	Engenharia de Software
GCS	Gerência de Configuração de Software
GNU GPL	GNU *General Public License*
IC	Item de Configuração
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
MPS.BR	Melhoria de Processos do Software Brasileiro
MRMPS	Modelo de Referências para Melhoria de Processos do Software Brasileiro
RC	*Request Change*
RM	*Request Maintenance*
RSS	*Really Simple Syndication*
SCM	Sistema de Controle de Mudanças
SCV	Sistema de Controle de Versão
SEI	*Software Engineering Institute*
SVN	Subversion
UML	*Unified Modeling Language*

1 Introdução

Projetos desenvolvidos em ambientes públicos como o projeto iNtegra, desenvolvido na Universidade Federal de Juiz de Fora, são projetos que têm como característica uma circulação intensa de pessoas interagindo e que precisam de uma documentação formal para apoio a execução de processos básicos visando melhorar o andamento do projeto. Com a evolução de um projeto, é natural o crescimento da quantidade de demanda que visam atender as necessidades dos diversos usuários e, com esse crescimento de demanda, as organizações buscam uma maturidade nos processos de software para atingir padrões de software de nível internacional. Os modelos de maturidade de software CMMI e MPS.Br citam a Gerência de Configuração de Software (GCS) como áreas chave para processos de manutenção de software mais maduros.

A GCS é uma área que está em constante evolução devido à necessidade de busca de métodos eficazes que permitam o desenvolvimento de um projeto de software de forma organizada. Esta área auxilia no propósito de estabelecer e manter a integridade dos produtos do projeto, isto é, ela consiste em identificar os itens de configuração, utilizar ferramentas para gestão, criação e liberação de *baselines*. *Baseline* (ou linha-base) tem por objetivo definir pontos no processo de desenvolvimento que gerenciam os itens de configuração e esses pontos podem ocorrer ao final de cada etapa do desenvolvimento ou não (FRANCO, 2010). A liberação de *baselines* envolve a aprovação de um conjunto de dados de configurações para o conjunto definido como itens de configurações. Além disso, a GCS visa controlar as mudanças dos itens de configuração, possibilitando a realização de auditorias provendo o status, e a configuração corrente para os interessados (DAFLON, 2005).

O controle de mudanças envolve o pedido de alterações em um software, podendo ser pedidos sobre correção de erros ou alguma mudança para a inserção de novos requisitos (PRESSMAN, 2006). A mudança em software envolve ainda a análise das requisições e a confirmação ou rejeição da mesma. Este controle de mudança pode ser feito através de ferramentas disponíveis que são classificadas como Sistemas de Controle de Mudanças (SCM).

O controle de versões necessita de uma unidade de desenvolvimento (Unidade de *Deployment*) que contém as versões finais do projeto com permissão de acesso para a equipe de desenvolvimento. Controlar versões também envolve operações de *check-out*

que é a atualização da versão de um desenvolvedor com a versão da unidade de desenvolvimento. Além das operações descritas, existe também confirmação das alterações (ou *commit*). Após a confirmação, o *check-in* é realizado, enviando o código-fonte de um repositório local para o repositório servidor.

Contextualizando estes conceitos com o projeto iNtegra, este trabalho apresenta uma proposta de processos que visam organizar o desenvolvimento do projeto, mantendo a integridade do produto e a melhora da qualidade do processo e do sistema. Portanto, o objetivo do trabalho é justificar a importância da aplicação de propostas para processos de gerência de configuração de software no ambiente do iNtegra.

Este trabalho apresenta os principais conceitos relacionados com a GCS e Qualidade de Software no capítulo 2, no capítulo 3 é apresentado um modelo do processo de gestão de mudanças e um para o processo de controle de versões e no capítulo 4 é descrito uma metodologia para implantação dos processos modelados no capítulo anterior sendo aplicados no projeto iNtegra. O capítulo 5 conclui o trabalho e propõe abordagens para trabalhos futuros.

2 Fundamentação Teórica

O desenvolvimento de um software não é apenas a construção de um sistema final imutável onde não ocorrerá nenhuma alteração depois que entregue ao cliente. Após a entrega, existe o processo de manutenção do software que realiza alterações com objetivo de aprimorar ou corrigir o sistema. Para isto, uma necessidade relacionada com o desenvolvimento de projetos de software é permitir a organização deste processo de desenvolvimento e manutenção, além do gerenciamento das alterações realizadas por várias pessoas. Esta necessidade é solucionada com a GCS.

A melhoria da qualidade de um software desenvolvido por uma equipe pode ser produzida através de aprimoramento dos processos desta equipe, isto é, baseando-se em modelos de qualidade como o CMMI (SEI, 2006) ou o MPS.BR (SOFTEX, 2007).

Contudo, todos esses processos devem ser devidamente documentados e isto pode ser feito através da definição de um plano de GCS em conformidade com as normas para elaboração de plano de GCS do IEEE Std 828 (IEEE, 2005).

2.1 Manutenção de Software

A manutenção de software é um processo geral de mudanças de um software depois que ele é entregue ao cliente (SOMMERVILLE, 2007). As mudanças feitas no software podem ser mudanças simples para corrigir erros de codificação, podem ser mudanças mais amplas que envolvem modificações em diversas partes e módulos dos sistemas, ou também melhorias com o objetivo de acrescentar novos requisitos (SOMMERVILLE, 2007). A manutenção de software pode ser justificada através de quatro atividades as quais serão discutidas a seguir.

A primeira atividade diz respeito ao reparo de defeitos de software sendo a manutenção corretiva. A correção de erros de codificação normalmente é menos custosa; os erros de projetos são mais caros, pois podem envolver a reescrita de vários componentes dos programas. Erros de requisitos são mais onerosos para serem reparados, pois pode ser necessário o reprojeto do sistema existente (SOMMERVILLE, 2007).

A segunda atividade ocorre por causa da rápida evolução característica da ciência da computação, por exemplo, novas gerações de hardware sendo anunciadas num curto intervalo de tempo constantemente; novos sistemas operacionais ou novos lançamentos

de versões antigas de componentes aparecem regularmente; equipamentos periféricos e outros elementos de sistema são frequentemente atualizados ou modificados, sendo a manutenção adaptativa. A vida útil dos aplicativos, por outro lado, pode facilmente ultrapassar 10 anos, prevendo um tempo de vida maior do que o ambiente de sistema para o qual foram originalmente desenvolvidos (PRESSMAN, 2006). Esta atividade de manutenção para adaptar o software a um ambiente operacional é conhecida como a manutenção adaptativa (SOMMERVILLE, 2007).

A terceira atividade é quando o projeto de software é bem-sucedido, sendo a manutenção perfectiva. À medida que o software é usado, recomendações de novos requisitos, de modificações em funções existentes e de ampliações gerais são recebidas dos usuários. Essa atividade é responsável pela maior parte de todo o esforço despendido em manutenção de software (PRESSMAN, 2006).

A quarta atividade é quando o software é modificado para melhorar a confiabilidade ou a manutenibilidade futura, ou para oferecer uma base melhor para futuros aprimoramentos, sendo a manutenção preventiva (PRESSMAN, 2006).

Na realidade, esta classificação entre atividades de manutenção não ficam completamente esclarecidos, por exemplo, quando o desenvolvedor modifica o sistema para ser executado em um novo ambiente ou sistema operacional, isso pode ser feito pela adição de novas funcionalidades para adaptação ao novo ambiente (SOMMERVILLE, 2007).

Estes tipos de manutenção citados anteriormente são bastante conhecidos, porém podem ter outros significados. Por exemplo, o termo manutenção corretiva é universalmente usado para se referir à manutenção de reparação de defeitos. Já a manutenção adaptativa algumas vezes se refere à adaptação a um novo ambiente e pode também significar adaptação do software aos novos requisitos. Finalmente, a manutenção evolutiva pode significar o aperfeiçoamento do software por meio da implementação de novas funcionalidades, mas em outras situações pode significar manter a funcionalidade do sistema, melhorando sua estrutura e seu desempenho (SOMMERVILLE, 2007).

Para contribuir na redução de custos na manutenção de software é necessário um código-fonte que seja padronizado e de fácil leitura. Porém, existem diversos fatores que levam a custos elevados quando uma aplicação sofre modificações:

1. Alta rotatividade da equipe de software: Depois que um sistema foi entregue, é normal a separação da equipe e que os integrantes da mesma se envolvam em

diferentes projetos. A equipe responsável pela manutenção dos sistemas irá necessitar de tempo longo e um esforço maior para compreensão do sistema antes de implementar as mudanças.

2. Responsabilidade contratual: O contrato para manter um sistema geralmente é separado do contrato de desenvolvimento deste, sendo que o segundo pode ser concedido a uma empresa diferente da que desenvolveu o sistema.
3. Experiência da equipe: Em muitos casos profissionais designados para realização da manutenção de software são inexperientes e sem familiaridade com o domínio da aplicação, pois ela não é vista como prioridade pelos engenheiros de software. Existe ainda a possibilidade de o software ter sido escrito em linguagens de programação obsoletas.
4. Idade e estrutura do programa: Com o passar do tempo as aplicações envelhecem, sua estrutura tende a se degradar pelas mudanças tornando-se cada vez mais difícil de ser compreendida e modificada. A documentação destas aplicações pode ser inexistente (ou ser inconsistente) e também as aplicações podem não ter sido submetidas ao gerenciamento de configuração, o que leva ao desperdício de tempo para encontrar as versões corretas dos componentes do sistema para a execução das mudanças

2.2 Gerência de Configuração de Software (GCS)

Segundo Shari, Hendler e Porter (2007) a configuração de um sistema de software é uma coleção de versões específicas de itens de configuração. Um item de configuração (IC ou *configuration item*) representa a agregação de hardware e software, sendo tratada pela GCS com um objeto único.

Segundo Dart (1991) a GCS é uma disciplina que visa controlar a evolução de software e propõe técnicas e ferramentas para a gestão da manutenção de software. A GCS tem por finalidade responder as seguintes questões básicas: "*Quando?*", "*Onde?*", "*Quem?*" e "*Por quê*?" as mudanças foram realizadas.

Vários são os motivos para estudar e adotar a GCS. Aplicações de qualquer porte necessitam do controle de versões, pois os engenheiros de software devem gerenciar o desenvolvimento de várias aplicações que estão sendo modificadas ao mesmo tempo e por equipes geograficamente distantes.

Outra motivação diz respeito às dependências que existem entre os componentes de software. Tais dependências muitas vezes são conhecidas localmente pelos desenvolvedores e analistas da aplicação. Desta forma, devem existir meios de compartilhar o conhecimento sobre estas dependências, a fim de facilitar a manutenção do software por outros desenvolvedores (SHAHRI, *et al.* 2007).

Assim como em outras áreas de estudo da Engenharia de Software (ES), a GCS necessita de soluções de software capazes de auxiliar na implementação de atividades, como a auditoria das mudanças realizadas, a definição de processos e análise estatística da evolução do software. Neste contexto, existem dois subsistemas que podem apoiar na GCS: Sistemas de Controle de Mudanças (SCM) e os Sistemas de Controle de versões (SCV). Este último foi objeto de estudo para o desenvolvimento deste trabalho e atendem especificamente à necessidade de se controlar o versionamento de um projeto de software.

Os principais conceitos inerentes à GCS são relacionados nas próximas seções.

2.2.1 Desenvolvimento Paralelo

O desenvolvimento paralelo se caracteriza por mais de uma pessoa participando de um mesmo projeto, mas modificando a aplicação de forma isolada e utilizando espaços de trabalho. Com isso, é possível efetuar a manutenção de software corretiva ou evolutiva de forma independente.

De acordo com Perry (apud APPLETON, 1998), muito dos problemas básicos no desenvolvimento paralelo podem ser resolvidos utilizando aplicações para o controle de versionamento dos itens de configuração estabelecidos em um plano de GCS. Um problema fundamental é a ausência de processos para o gerenciamento e controle de várias mudanças de forma paralela aproveitando e alocando mais de um desenvolvedor para resolver a manutenção na aplicação.

2.2.2 Aprovação formal de novas versões (*Baselines*)

As *baselines* representam conjuntos de itens de configuração formalmente aprovados que servem como referência para as próximas etapas do desenvolvimento (DANTAS, 2008). Para uma compreensão melhor do que é uma *baseline* (ou linha-base) é necessária a introdução do conceito de *baselevel* que é uma configuração de um projeto auto-suficiente para servir como base estável para desenvolvimento. Uma *baseline* é uma *baselevel* apropriada para uma versão formal interna ou externa (APPLETON, 1998).

Liberar uma *baseline* envolve aprovar um conjunto de dados de configurações para o conjunto definido como itens de configurações, a partir do sistema de gerenciamento de configurações, tornando o software disponível para manutenções futuras e o desenvolvimento posterior (CHRISSIS, 2002).

Dentro do contexto da engenharia de software, pode-se definir uma *baseline* como uma referência no desenvolvimento de um software, que é caracterizado pela entrega de um ou mais itens de configuração (IC) e pela aprovação desses IC obtido por meio de uma revisão técnica (PRESSMAN, 2006) ou ainda a *baseline* é uma especificação ou um produto que foi formalmente revisado e aprovado que serve de base para o desenvolvimento posterior e só deverá ser modificado através de procedimentos formais de controle de mudanças (IEEE, 1996).

A *baseline* nos ajuda a controlar todas as mudanças realizadas no projeto sem impedir as mudanças justificáveis (PRESSMAN, 2006). Para ilustrar este conceito, imaginemos um protótipo de interface que modela o comportamento de um sistema de gerenciamento acadêmico, assim que este protótipo for analisado, revisado e aprovado com fundamentos técnicos após eventuais correções, este protótipo torna-se uma *baseline* aonde todas as alterações adicionais posteriores terão a mesma como referência. Para efeitos de aplicação de conceitos de engenharia de software, assim que um IC se transforma numa *baseline* ele deve ser enviado ao repositório para que todos os integrantes da equipe possam ter acesso ao IC. Esta comunicação de arquivos entre repositório e a máquina do desenvolvedor devem ser feita através de sistemas de controle de versão que possuem os recursos necessários e que mantêm uma organização no projeto. Um recurso que exemplifica a necessidade de uma ferramenta do tipo (SCV) é quando se pretende garantir a integridade do arquivo que se encontra no repositório, por exemplo, a *baseline* pode realizar o travamento (*lock*) aonde os outros desenvolvedores não podem alterar o arquivo do repositório e sim devem realizar a cópia para sua máquina e trabalhar independentemente do que se encontra no repositório.

2.2.3 Ramificações ou *Branching*

A GCS permite o desenvolvimento paralelo de novas funcionalidades por uma equipe ou várias equipes de software, de forma isolada e independente utilizando *branch*, o qual permite isolar o desenvolvimento destas modificações (DANTAS, 2008). *Branch*

(ou *branching*, ramificação) é um mecanismo amplamente utilizado por muitas ferramentas de controle de versão que suporta desenvolvimento concorrente.

Normalmente, na implementação de várias aplicações cria-se um *branch* que será a linha principal de desenvolvimento. Nesta linha principal ocorrem as atualizações de manutenções evolutivas ou inserção de novos requisitos para geração de *baselines* e versões estáveis da aplicação. Neste cenário, é comum os desenvolvedores atualizarem a linha principal de desenvolvimento diariamente, enquanto a última versão do sistema se encontra em produção e sendo utilizada pelo usuário. Porém, a aplicação em produção pode apresentar erros em requisitos mal-elaborados, erros apresentados na própria sintaxe da linguagem de programação, impedindo a utilização e funcionamento completo da aplicação.

Diante deste cenário, o usuário relata o erro à equipe de desenvolvimento que continuará modificando a aplicação por meio do *branch* da linha principal. Outra equipe de desenvolvedores poderá criar um *branch* da versão que apresentou erro em produção, copiando os itens de configuração por meio de um comando *checkout*, e efetuar as modificações, gerando uma nova versão da aplicação (APPLETON, 1998).

Uma estratégia bastante utilizada, quando ocorre o término destas correções de *bug*, é unir os itens modificados em um *branch* de correção com a linha principal de desenvolvimento. Neste caso, são executadas duas operações básicas oferecidas pelas ferramentas de controle de versão. A primeira é a execução de comando de *checkin*, e outra é a execução de uma operação de merge ou junção dos itens modificados de um *branch* em outro.

Outras definições também devem ser relacionadas sobre *branch*. As *codelines* ou linhas de desenvolvimento são designadas para cada projeto e são compartilhadas por vários desenvolvedores. A primeira linha de desenvolvimento definida no projeto é, por convenção, nomeada *mainline*. O ramo é criado no repositório e representa uma linha secundária de desenvolvimento que pode ser unida novamente à linha principal (*mainline*) por meio da operação de junção (*merge*), como foi descrito anteriormente. Atualmente, a necessidade de atender, ao mesmo tempo, as múltiplas demandas do projeto torna o uso de ramos um diferencial (DANTAS, 2008).

2.2.4 Estratégias para criação de *branch*

Estratégias para criação de *branch* buscam fornecer padrões para decompor um *workflow* de um projeto em linhas separadas de desenvolvimento e depois recompor essas linhas para o fluxo principal de trabalho (APPLETON, 1998). As estratégias podem se adequar de acordo com os clientes ou plataformas, permitindo a customização do desenvolvimento do sistema. Uma seleção correta da estratégia permite (MURTA, 2011):

- Desenvolvimento/manutenção em paralelo de liberações
- Isolamento entre desenvolvedores e equipes
- Identificação das requisições de modificação que pertencem a cada *baseline*
- Adição ou remoção de uma requisição de modificação em uma *baseline*
- Customização do sistema para diferentes clientes ou plataformas
- Manutenção da integridade de *baseline*

As estratégias de *branching* podem ser do seguinte tipo:

- Manutenção: caótica, em série e em cascata.
- Organização: por desenvolvedores, por subprojetos (componentes da arquitetura), por requisições e por customizações.
- Verificação: contínua, periódica e pré-liberação.

A manutenção caótica (Figura 2.1) é a estratégia de ramificação onde não existe a possibilidade de separação do que é manutenção corretiva de manutenção evolutiva (MURTA, 2011), a evolução e a correção do software ocorrem na *baseline,* não existindo ramificações (DANTAS, 2008).

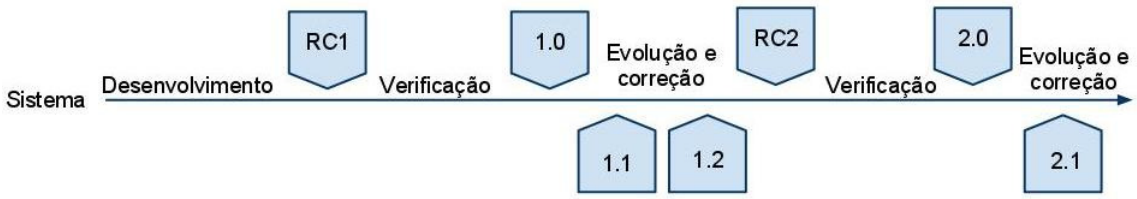

Figura 2.1- Manutenção caótica (MURTA, 2011)

A estratégia em manutenção em série (Figura 2.2) é separar as evoluções das correções no software que permite ser usada quando uma versão *release* do produto será

entregue para a homologação (DANTAS, 2008). Neste caso, o ramo principal fica com a evolução e os ramos auxiliares com as correções.

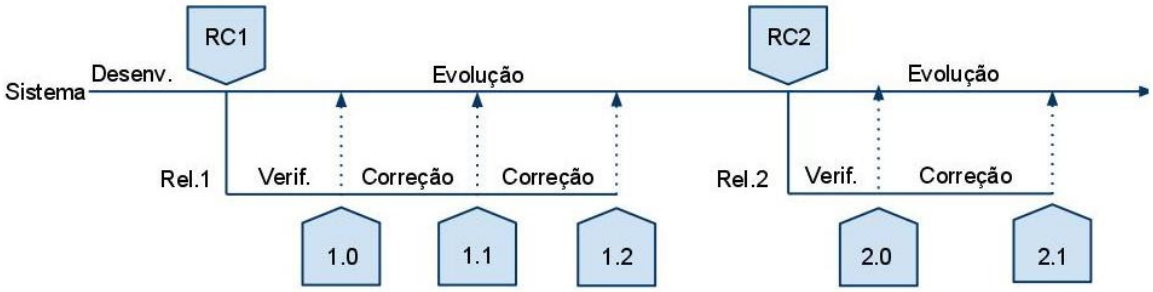

Figura 2.2 – Manutenção em Série (MURTA, 2011)

A manutenção em cascata (Figura 2.3) possui o mesmo princípio que a manutenção em série, porém no ramo principal ocorrem as correções e no auxiliar a evolução.

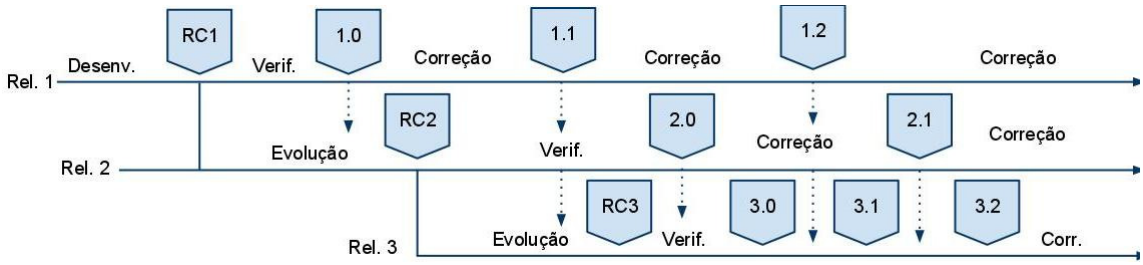

Figura 2.3 – Manutenção em cascata (MURTA, 2011)

A estratégia de ramificação de organização por desenvolvedores (Figura 2.4) permite que cada desenvolvedor faça *check-ins* intermediários sem a necessidade de interferência nas ramificações dos outros desenvolvedores. No ramo principal é a integração e os ramos auxiliares são dos desenvolvedores.

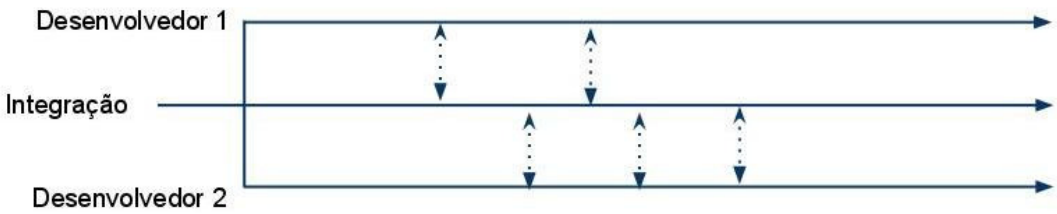

Figura 2.4 – Ramificação de organização por desenvolvedores (MURTA, 2011)

Na organização por subprojetos ou componentes da arquitetura (Figura 2.5), o ramo principal é da integração e os ramos auxiliares dos projetos, fornecendo um baixo isolamento entre membros de um mesmo subprojeto, mas isola-se a equipe do subprojeto das demais.

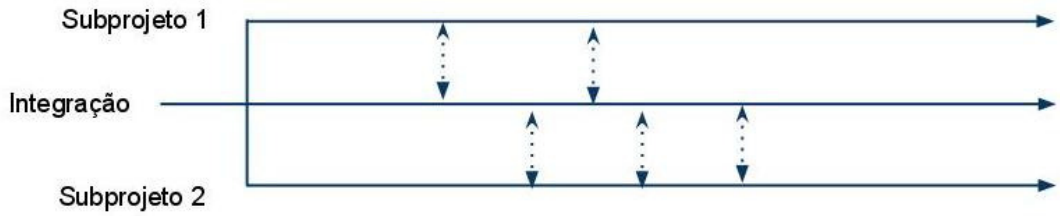

Figura 2.5 – Ramificação de organização por subprojeto (MURTA, 2011)

A organização por requisições (Figura 2.6) torna possível a remoção de uma requisição do produto que permite a identificação de cada requisição (*change sets*). No ramo principal ocorre a integração e os ramos auxiliares, as requisições.

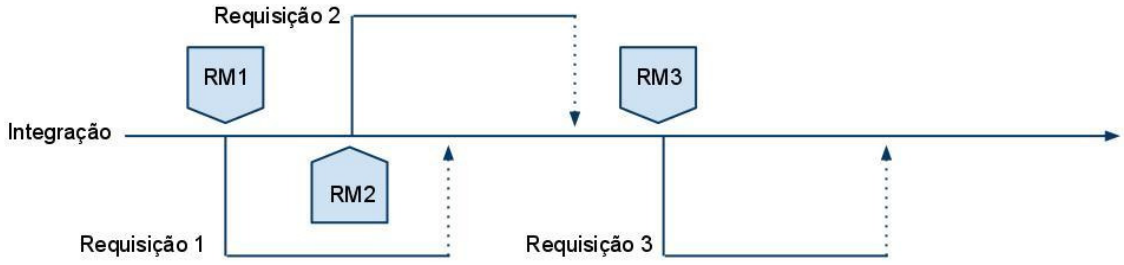

Figura 2.6 – Ramificação de organização por requisições (MURTA, 2011)

A estratégia de ramificação de organização por customizações (Figura 2.7) é adequada para quando a seleção de variabilidade em tempo de construção é mais apropriada. No ramo principal ocorre a integração e os ramos auxiliares, customizações.

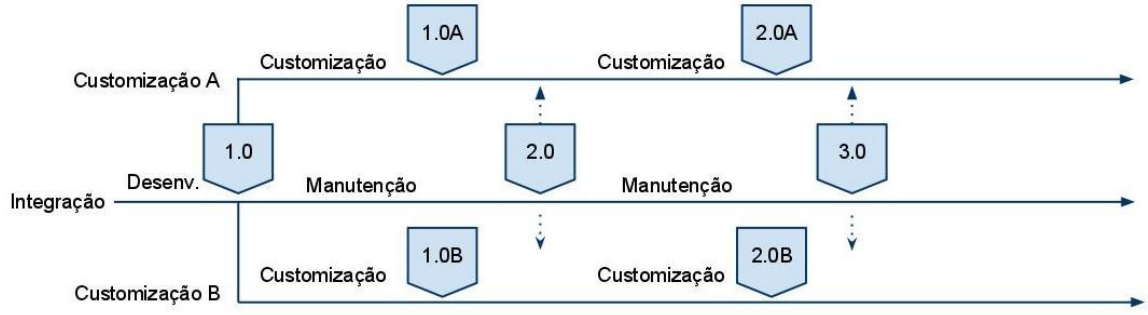

Figura 2.7 – Ramificação de organização por customização (MURTA, 2011)

A estratégia por verificação contínua é quando toda e qualquer integração só é efetuada após passar pela verificação. Uma estratégia de estado seguro e que possui um alto custo em caso de verificação não automatizada. O ramo principal sempre pronto para liberação. A figura 2.8 ilustra a estratégia de ramificação de verificação contínua.

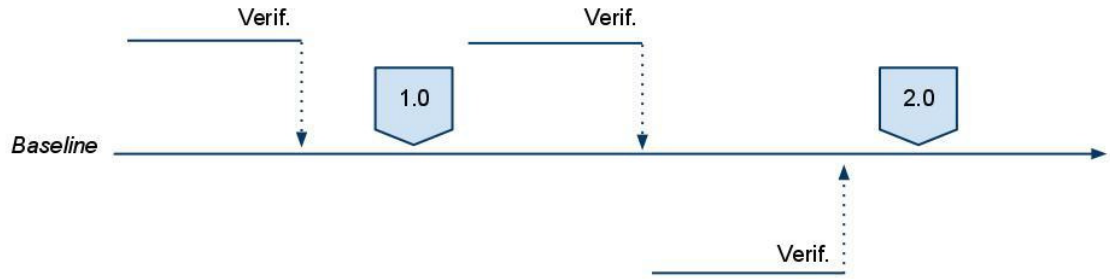

Figura 2.8 – Ramificação por verificação contínua (MURTA, 2011)

Na ramificação por verificação periódica, as integrações são feitas de forma não sincronizadas com verificações, e as verificações são executadas de tempos em tempos, isto torna independente a atividade de verificação. Utiliza-se um rótulo para demarcar pontos já verificados e possui um custo médio em caso de verificação não automatizada (Figura 2.9).

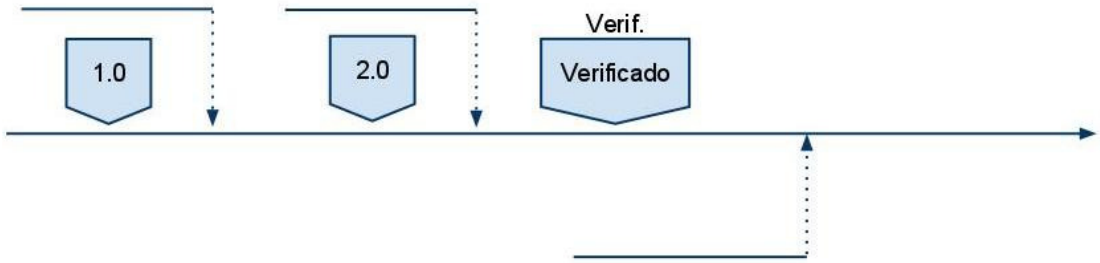

Figura 2.9 – Ramificação por verificação periódica (MURTA, 2011)

Na estratégia de ramificação pré-verificação, a verificação é executada somente antes da liberação (Figura 2.10). Possui um efeito de "congelamento" (ou travamento) do ramo principal que só aceita correções durante congelamento. Esta estratégia tem um custo baixo em caso de verificação não automatizada.

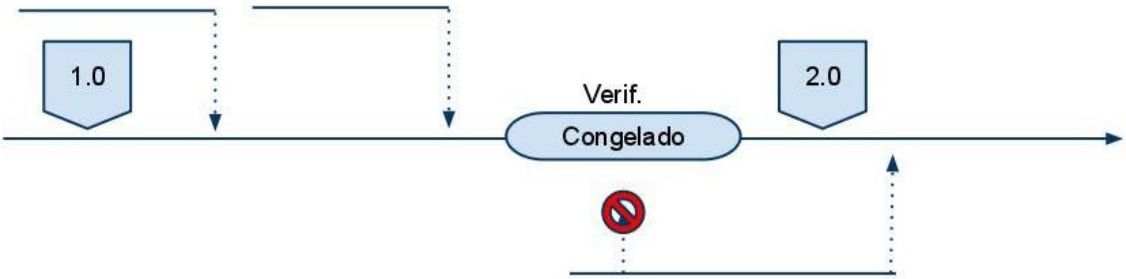

Figura 2.10 - Ramificação pré-verificação (MURTA, 2011)

2.2.5 Ferramentas de Controle de Mudança

O controle de mudanças combina procedimentos humanos com ferramentas automatizadas para proporcionar um mecanismo de controle de mudanças que pretende evitar mudanças descontroladas que levam rapidamente ao caos. O processo de controle de mudanças é o processo de solicitação, tomada de decisão, planejamento, implementação e avaliação das alterações ao sistema (PRESSMAN, 2006).

Para apoiar a execução das atividades de GCS, é importante adotar ferramentas apropriadas para o registro e monitoramento das atividades. Basicamente, as ferramentas para GCS incluem sistemas para manter as modificações nas bibliotecas de software de forma separada, e um sistema de arquivos para gerenciar e registrar as operações de cópia e o envio dos artefatos modificados localmente pelos desenvolvedores.

2.2.5.1 Trac

O Trac (EDGEWALL SOFTWARE, 2007) é um sistema *Open Source* sendo uma ferramenta para gerência de mudanças que possui uma interface web e suporte para integração com ferramentas de controle de versões. Ele permite como base de dados para persistência as solicitações SQLLite, PostgreSQL ou MySQL. O código-fonte se encontra na linguagem Python (EDGEWALL SOFTWARE, 2007).

Trac oferece suporte a notificações por E-mail ou *Really Simple Syndication* (RSS), permite customização do *workflow* de controle de solicitações, tendo como *workflow* padrão o modelo descrito na Figura 2.11, e possui ainda recursos de visualização geral dos tickets, dados estatísticos de cada ticket do sistema, *plug-in* para criação de matriz de rastreabilidade e um módulo de criação de relatórios. Os relatórios são baseados em consultas do tipo SQL SELECT e convenções de nomes que permitem grande flexibilidade e formatações tais como formatações de cores e quebras de linha. Ele segue o formato *wiki* e suporta marcações especiais para criação de ligações entre macros e designações de tarefas (EDGEWALL SOFTWARE, 2007).

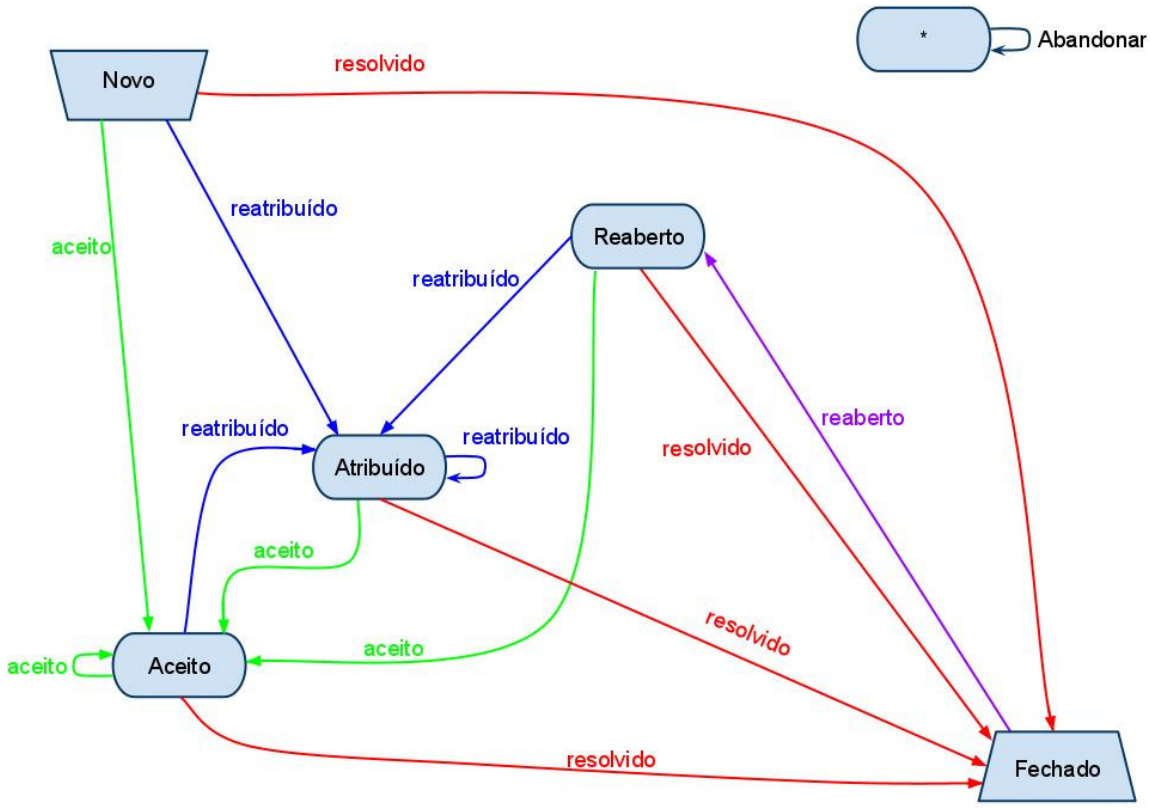

Figura 2.11 – *Workflow* padrão do Trac (EDGEWALL SOFTWARE, 2007)

2.2.5.2 Mantis

O Mantis (MANTIS, 2009) é uma ferramenta de gerência de mudanças *Open Source* que atualmente suporta como base de dados MySQL, MS SQL e PostGreSQL, com uma versão experimental na base ORACLE. O Mantis funciona a partir de qualquer plataforma que suporte PHP, oferecendo também uma interface Web com notificações por E-MAIL ou RSS. O *workflow* do Mantis também é customizável e tem o modelo padrão descrito na Figura 2.12, sendo os tickets identificados como requisições ou "*issues*".

O acompanhamento das mudanças que são feitas por um projeto integrado com o Mantis pode ser feita através de um *log* (registro com as alterações) de mudanças ou gráfico de relações entre tarefas, permitindo um controle de rastreabilidade do projeto.

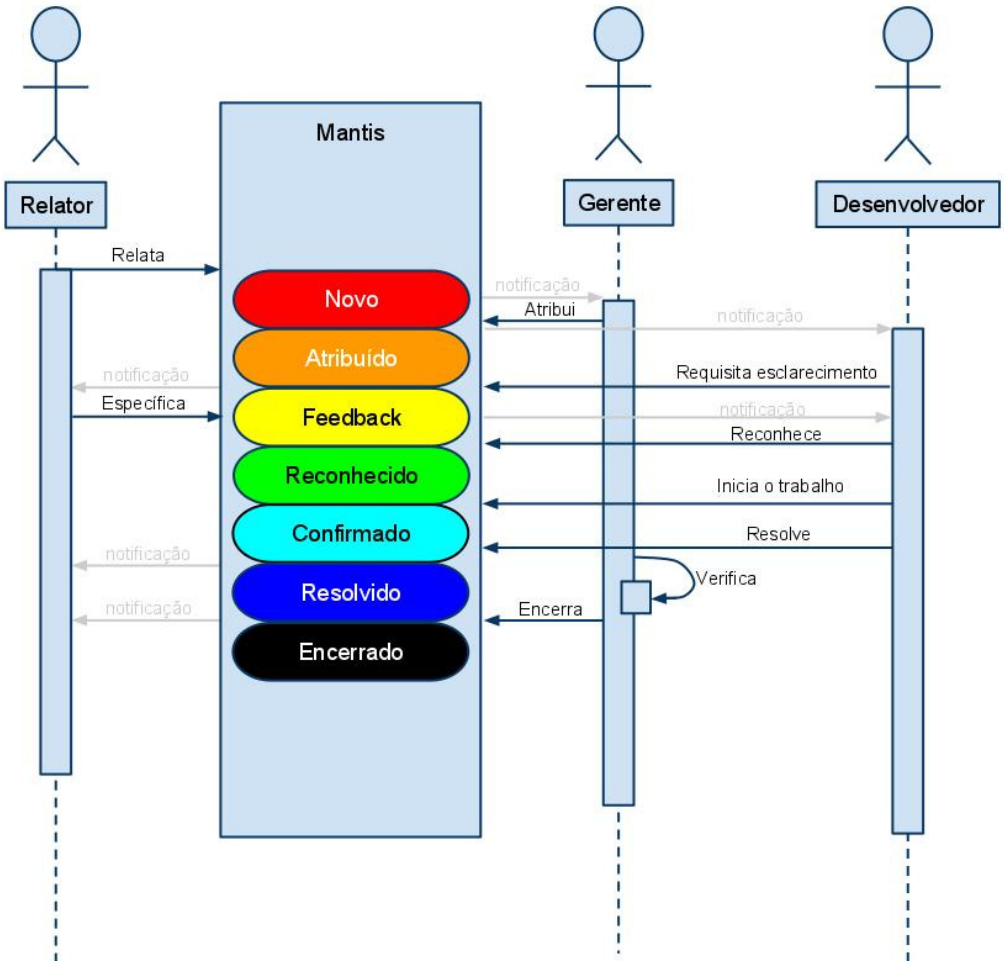

Figura 2.12 - *Workflow* padrão do Mantis – (WARE LAB, 2011)

2.2.5.3 Bugzilla

O Bugzilla (THE BUGZILLA TEAM, 2011) é uma sistema *Open Source* que atualmente suporta como base de dados MySQL, PostgreSQL e ORACLE. As notificações do Bugzilla são descritas como "*bugs*" e a lista destes pode ser visualizada numa interface Web, Atom, iCalendar, CSV para importação e *spreadsheets*. O Bugzilla permite envios de e-mail para um endereço de sua configuração que interage (cria ou altera) um *bug*. O Bugzilla também possui recursos para customização de *workflow*, sendo o *workflow* padrão descrito na Figura 2.13.

Como se pode notar no modelo, há diversas situações de fechamento possíveis de um bug dentre os quais podem ser “corrigido”, “duplicado”, “naocorrige”, “funcionacomigo” e “inválido” (Traduzidos para o português).

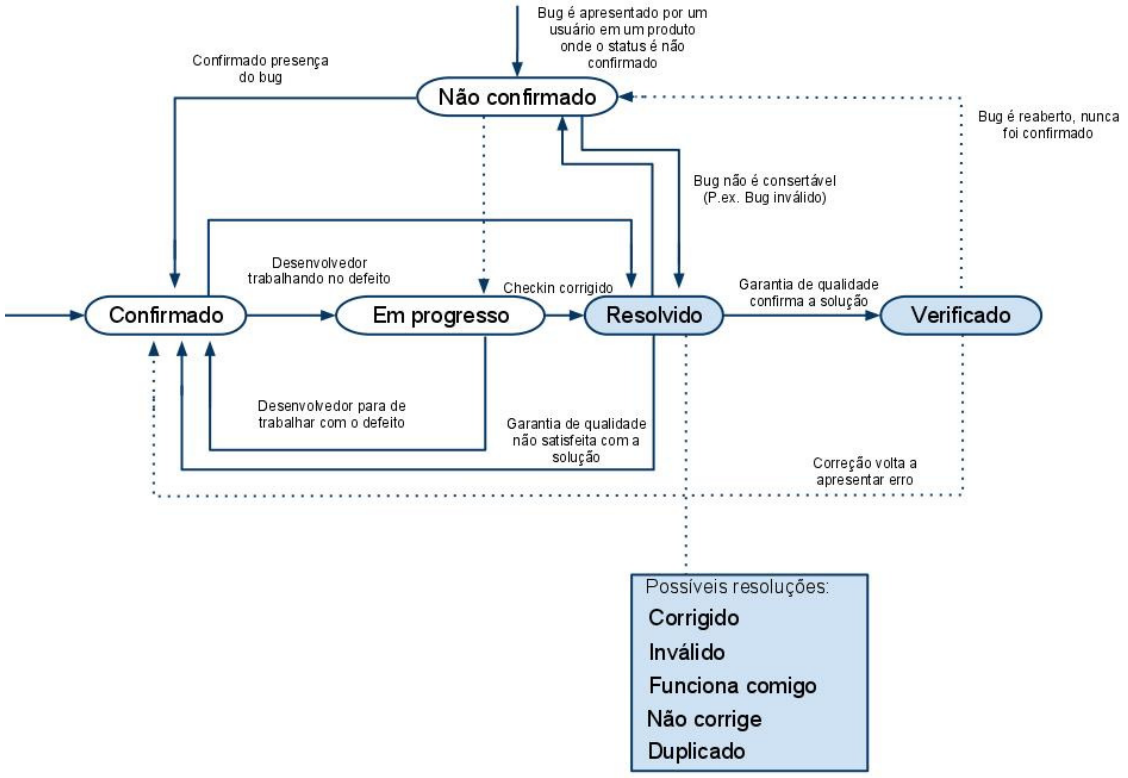

Figura 2.13 - *Workflow* padrão do Bugzilla - (THE BUGZILLA TEAM, 2011)

2.2.6 Ferramentas de Controle de Versão

O controle de versão combina procedimentos e ferramentas para gerenciar as diferentes versões de itens de configuração que são criadas durante o desenvolvimento do projeto (PRESSMAN, 2006). As ferramentas de controle de versão (ou sistemas de controle de versão) têm como objetivo gerenciar as versões, realizar operações de cópia e o envio e armazenamento de versões dos itens de configuração do projeto.

2.2.6.1 Subversion

O Subversion (COLLINS-SUSSMAN *et al.*, 2006) foi criado para rodar numa camada portável chamada "*the Apache Portable Runtime library*" (APR), isto é, ele deve funcionar em qualquer sistema operacional em que o servidor "Apache httpd" é executado: Windows, Linux, todos os sabores (*flavors*) do BSD, Mac OS X, Netware (COLLINS-SUSSMAN *et al.*, 2006).

As principais funções básicas de sistema de controle versão como *check-in* (atômicos ou não), *check-out*, *lock*, *branch*, *tag*, *diff*, *merge* e *history*, estão incluídas no SVN. Os comandos de *move*, *rename* e *copy* só registra o histórico caso o repositório

destino seja o mesmo do repositório fonte do Subversion (é possível a criação de repositórios distribuídos).

Existe a possibilidade também de criação de um "*hook script*" para serem executados antes e depois da chamada de comandos do Subversion (COLLINS-SUSSMAN *et al.*, 2006).

Para o controle de concorrência, o Subversion usa a solução definida como "*The Copy-Modify-Merge*" que consiste em manter um arquivo no repositório e copiar este arquivo para os usuários que desejam ler e alterar (COLLINS-SUSSMAN *et al.*, 2006).

2.2.6.2 **Mercurial**

O Mercurial (O'SULLIVAN, 2009) é um software sob licença GNU GPL v2 que tem compatibilidade com os sistemas operacionais Unix-like, Windows, Mac OS X. Uma das recomendações para instalações no Windows é a versão "TortoiseHG" que é uma versão sem dependências externas, do tipo "apenas funciona" e que inclui uma interface gráfica. O Mercurial tem suporte as principais funções básicas de um sistema de controle de versão: *annotate*, *branch*, *commits* (atômicos ou não), *diff*, *merge*, *log* (*history*), *tag*, *update* e *lock* (O'SULLIVAN, 2009).

Um dos diferenciais do Mercurial é o suporte a ferramentas como TortoiseMerge ou KDiff3 para realização das comparações de código entre dois (ou até três) arquivos e possível realização da junção, caso nenhum programa esteja associado, o padrão do merge do Mercurial é baseado no clássico "*3-way Merge*". O Mercurial permite a execução de scripts do tipo *hooks pre--<command>* e *post--<command>* que se comportam de acordo com as invocações do comando associado (O'SULLIVAN, 2009).

2.2.6.3 GIT

Para instalação do GIT (GREAVES, 2011), é necessário um dos sistemas operacionais Debian/Ubuntu, Windows, OS X.O GIT tem os principais comandos básicos de um sistema de controle versão como *check-out*, *commit*, *branch*, *tag*, *diff*, *merge*, *log* e *notes* (ou *annotation*). O Git mantém o registro de todos os *commits* que são realizados durante um projeto, podendo ser restaurando um ponto anteriormente com bastante agilidade e ainda é capaz de manter o histórico após realizada uma alteração, por exemplo, da execução do comando "git-mv" (*move*) (GREAVES, 2011).

Um dos principais destaques do GIT é que ele é rápido, escalável, distribuído que permite operações de alto-nível e total acesso para os usuários.

A possibilidade de Hooks é limitada por alguns comandos através do GIT sendo possível executar nos momentos "*applypatch-msg*", "*pre-applypatch*", "*post-applypatch*", "*pre-commit*", "*prepare-commit-msg*", "*commit-msg*", "*post-commit*", "*pre-rebase*", "*post-checkout*", "*post-merge*", "*pre-receive*", "*update*", "*post-receive*", "*post-update*", "*pre-auto-gc*" e "*post-rewrite*" (GREAVES, 2011).

2.3 Qualidade de Software

Segundo Weinberg (apud KOSCIANSKI *et al.*, 2007), a qualidade é relativa, o que é qualidade para uma pessoa pode ser falta de qualidade para outra. Uma definição citada por Crosby (apud KOSCIANSKI *et al.*, 2007) cita que a qualidade é conformidade aos requisitos o que tenta citar que a qualidade pode ser definida que o produto pode ser classificado se ele atende seus requisitos de maneira esperada. Esta definição de qualidade ainda não está completa, pois existem três fatos que perturbam esse conceito.

O primeiro fato é deve-se levar em consideração as características observadas no produto e as características que foram especificadas para sua construção. O segundo deve-se lembrar que existem várias fontes de erros que podem corromper os dados utilizados para caracterizar um produto. O último fato a considerar é o papel de diferentes clientes em um mesmo projeto, onde os requisitos foram definidos por alguém, logo a qualidade depende das escolhas que alguém efetuou. Para o caso de qualidade de software especificamente, podemos ver que este último ponto ilustra que diferentes *stakeholders* têm em mente diferentes requisitos que visam atingir diferentes objetivos e ainda expressá-los de maneiras distintas. O termo *stakeholders* se refere as pessoas ou grupos que são afetados pelo sistema de forma direta ou indireta (SOMMERVILLE, 2007).

É importante ter em consciência que utilizar um modelo de qualidade para desenvolvimento de projetos não é garantia para uma manutenção de software com qualidade. A qualidade de software ainda da comunicação mais próxima entre desenvolvedores e também entre engenheiros de requisitos e clientes, ainda que existam ferramentas para apoiar nesses procedimentos a qualidade do processo de levantamento de requisitos interfere muito na qualidade final do produto de software (KOSCIANSKI, 2007).

De forma mais objetiva, para conseguir qualidade no desenvolvimento e produto de software é importante enfatizar aspectos como: requisitos de software pois são a base

a partir da qual a qualidade é medida, padrões especificados definem um conjunto de critérios de desenvolvimento que orientam a maneira segundo a qual o software passa pelo trabalho de engenharia e um conjunto de requisitos implícitos ou não-funcionais (PRESSMAN, 2006).

Mesmo ao definir processos para atingir a qualidade necessária para o desenvolvimento de um software com qualidade, deparamos com o desafio de manter esta qualidade durante a manutenção deste produto. Os modelos de qualidade para melhoria do processo de desenvolvimento propõem processos para o desenvolvimento do software, mas a manutenção de software também pode ser considerada como parte do processo de desenvolvimento.

A qualidade alcançada ao término do processo de desenvolvimento não garante necessariamente que a mesma irá se perpetuar durante toda a vida útil deste software. Deve-se também estabelecer processos que visam garantir a qualidade durante o ciclo de manutenção, para uma manutenção preventiva, o processo de desenvolvimento que visa assegurar a qualidade pode ser realizado da mesma maneira de desenvolvimento de um novo produto. Os demais tipos de manutenção de software requerem processos bem definidos que variam de acordo com as características da equipe e do projeto.

2.3.1 CMMI

O SEI (*Software Engineering Institute*- Instituto de Engenharia de Software) (SEI, 2006) é um instituto criado para aumentar as capacitações da indústria de software nos EUA. Na década de 1980, o SEI iniciou um estudo de formas de avaliação de capacitação de fornecedores que teve como resultado o Modelo de Maturidade de Capacitação (CMM – *Cappability Maturity Model*), que exerceu uma influência na comunidade de engenharia de software em considerar o aprimoramento de processos (SOMMERVILLE, 2007). Após o desenvolvimento de diversos modelos de maturidade de processos semelhantes por diferentes organizações, numa tentativa de integrar a grande quantidade destes modelos o SEI embarcou no novo programa para desenvolver um Modelo de Maturidade de Capacitação Integrada, o CMMI - *Cappability Maturity Model Integration*, que substitui os CMMs de Engenharia de Sistemas e o de Software e integra outros modelos de engenharia (SOMMERVILLE, 2007).

Em 2002, surgiu então o CMMI com o objetivo de ser um framework de aprimoramento de processos que tem aplicabilidade ampla por meio de uma variedade de

empresas. No CMMI são definidos dois modelos: o modelo contínuo e o modelo por estágios (ou versões). A versão por estágios do modelo CMMI é compatível com o CMM para Software e permite aos processos de desenvolvimento e gerenciamento de sistemas de uma organização serem avaliados e que recebam um nível de maturidade de 1 a 5. A versão contínua do modelo CMMI permite uma classificação mais fina e avalia 24 áreas de processo em uma escala de 1 a 6 (SOMMERVILLE, 2007).

Entre as 24 áreas de processo relevantes para a capacitação e aprimoramento identificadas pelo modelo CMMI as Gerência da Qualidade de Processo e a Gerência de Configuração de Software encontram-se definidas como áreas de apoio servindo de base para suporte e aquisição dos próximos níveis do CMMI (SOMMERVILLE, 2007).

2.3.2 MPS.BR

O MPS.BR (SOFTEX, 2007) surgiu com o propósito de contribuir para a melhoria da qualidade de software dentro da realidade das micro, pequenas e médias empresas de software brasileiras, que possuem poucos recursos e que necessitam obter melhorias significativas nos seus processos de software. O modelo adota processos que podem ser implementados em um espaço curto de tempo, inserindo estas empresas em um mercado mais competitivo (SOFTEX, 2007).

Os processos propostos pelo MPS.BR também podem apoiar outros grupos de empresas. Espera-se que ele seja adequado ao perfil de empresas com diferentes tamanhos e características, públicas e privadas. Baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços correlatos, e se encontra descrito em documento no formato de guias (SOFTEX, 2007).

O Guia de Implementação fornece orientações para implementar nas organizações os níveis de maturidade descritos no Modelo de Referência MRMPS, detalhando os processos contemplados nos respectivos níveis de maturidade.

Os níveis de maturidade descritos pelo guia são divididos em sete níveis, começando do nível G até o nível A. O resumo em questão apresenta como as empresas podem evoluir do nível G para o F. O principal foco do nível F é agregar processos que apoiarão a gerência do projeto de software no que diz respeito à Garantia da Qualidade e Medição, bem como aqueles que irão organizar os artefatos de trabalho por meio da Gerência de Configuração. Esses processos adicionais vão possibilitar uma maior

visibilidade de como os artefatos são produzidos nas várias etapas do projeto e do processo (SOFTEX, 2007).

Entre os processos propostos para que uma empresa possa migrar do nível G para o F, encontram-se os processos de Aquisição, Gerência de Configuração, Garantia da Qualidade e Medição. O propósito do processo Aquisição é gerenciar a aquisição de produtos e/ou serviços que satisfaçam a necessidade expressa pelo adquirente. Em síntese, o processo se inicia com a identificação de uma necessidade do cliente e encerra com a aceitação do produto e/ou serviço, e tem como objetivo selecionar um fornecedor e garantir que o fornecedor entregue o produto conforme definido.

Em relação à Gerência de Configuração os processos são estabelecidos visando manter a integridade de todos os produtos de trabalho de um processo ou projeto, e disponibilizá-los a todos os envolvidos. Entre as atividades envolvidas no processo de Gerência de Configuração é possível citar: a identificação de itens de configuração, o controle do acompanhamento da evolução dos itens e a geração de estatísticas visando a melhoria do processo (SOFTEX, 2007).

2.4 Normas para elaboração de Planos de Gerenciamento da Configuração de Software

A norma IEEE Std 828 (IEEE, 2005) estabelece o conteúdo mínimo exigido para um plano de GCS. Esta norma visa controlar a evolução de sistemas de software e propõe técnicas e ferramentas para a gestão da manutenção de software, além de descrever como engenheiros de software podem elaborar um planejamento para GCS, a fim de estabelecer processos para gestão de mudanças, definição da configuração do software, identificação de IC, monitoramento e auditoria do plano de GCS.

Um bom planejamento resulta num documento que captura as informações de planejamento, realiza a informação da propriedade do projeto, comunica a todos quem são os afetados e fornece o básico para o planejamento em curso (IEEE, 2005). As informações do planejamento da GCS devem obedecer a restrições fundamentais como deve existir um formato de documento definido, este documento deve conter todas as informações do planejamento (por inclusão ou referência) e um documento com o título "Plano de gerência de configuração de software" deve existir tanto numa forma-única ou anexado com outro documento do projeto.

Segundo a norma IEEE Std 828 (2005), o plano de GCS e suas informações devem estar divididos em seis classes, seguindo a esta mesma seqüencia a menos que esteja especificado um formato diferente na introdução:

1. Introdução: Descreve o propósito do plano, escopo da aplicação, termos-chave e referências.
2. Gestão da GCS: Identifica as responsabilidades e autoridades para gerenciamento e acompanhamento das atividades de GCS planejadas.
3. Atividades da GCS: Identifica todas as atividades a serem realizadas na aplicação do projeto.
4. Agendamento das atividades de GCS: Identifica a coordenação das atividades de GCS exigidas com outras atividades do projeto.
5. Recursos utilizados para GCS: Identifica ferramentas e recursos físicos e humanos necessários para a execução do plano.
6. Manutenção do plano da GCS: Identifica como plano será mantido enquanto estiver em vigor.

A seção de introdução do padrão visa fornecer uma visão simplificada e global das atividades de GCS devendo incluir os tópicos propósito do plano, escopo, definição dos termos chaves e referências.

O propósito do plano deve brevemente abordar o porquê da existência do plano e descrever qual é o público alvo. O escopo deve abordar a aplicabilidade da GCS, limitações e as suposições em que o plano é baseado. O tópico sobre a definição dos termos chaves define são estes termos e como eles se aplicam ao plano. As referências de todos os documentos relacionados com o plano devem estar unicamente identificadas no tópico referências.

A classe “Gestão da GCS” descreve a designação de responsabilidade e autoridades para realização das atividades da GCS e suas gestões. Ela deve ser composta de quatro tópicos: a organização (ou organizações) do projeto as quais a GCS se aplica, as responsabilidades destas organizações, referências para políticas e diretivas de GCS que se aplicam ao projeto e a gestão dos processos de GCS.

A seção de atividade da GCS identifica todas as funções e tarefas exigidas para gerenciar a configuração do software como especificado no escopo do plano na seção de introdução. Devem ser identificadas as atividades técnicas e administrativas da GCS.

Esta seção é uma das mais importantes do plano que é tradicionalmente agrupada em cinco funções: identificação da configuração, controle da configuração, geração de estatísticas (relato de status), auditoria do processo de gestão de configuração e gerenciamento da liberação e entrega.

A identificação da configuração inclui identificação de nome e descrições dos documentos fisicamente e características funcionais do código, especificações, design e elementos de dado a serem controlados para o projeto (IEEE, 2005).

O controle da configuração envolve atividades para solicitação, avaliação, aprovação ou reprovação e implementação de mudanças na *baseline* dos itens de configuração. O controle de configuração também se aplica ao processamento dos pedidos para desvios e renúncias para disposições das especificações do contrato de adquirinte-fornecedor e definição das informações a serem usadas para rastreamento e documentação dos IC, procedimentos para solicitações de mudanças, avaliação das mudanças, atividades para verificação e implementação das mudanças aprovadas (IEEE, 2005).

A geração de estatísticas avalia os status dos componentes e dos pedidos de mudanças para geração de relatórios que fornecem informações sobre a freqüência em que um software é modificado. O plano deve incluir as seguintes informações: quais elementos de dados e métricas de GCS serão rastreados e relatados para *baselines* e mudanças, que tipos de relatórios estatísticos serão gerados e suas freqüências com as informações devem ser coletadas, armazenadas, processadas, relatadas e protegidas de perdas e como o acesso aos dados de status devem ser controlados.

Auditoria do processo de gestão de configuração tem como o objetivo a validação da completude de um produto e da integridade entre os componentes de software, basicamente esta atividade consiste na realização de auditorias que determinam a extensão em que o atual IC reflete a características físicas e funcionais exigidas. O plano deve identificar a auditoria e revisão de configuração a ser realizada para o projeto, e definir também o objetivo de uma auditoria, os IC sob auditoria ou revisão, os participantes por cargo, documentação exigida para estar disponível para auditoria, o procedimento para registro de quaisquer atitudes que alteram a *baseline* e ações específicas a ocorrer após a aprovação. No mínimo, uma auditoria de configuração deve ser realizada previamente a liberação de um IC.

A gerência da liberação e da entrega descreve como os produtos de software foram construídos, liberados e entregues, e como a documentação será formalmente controlada. Procedimentos para acomodar desvios aprovados e renúncias deverão ser incluídos nos mecanismos de controles. Cópias principais do código e documentação devem ser mantidas durante a vida do produto de *software*. O código e documentação que contêm informações de segurança ou funções críticas de seguranças devem ser manuseados, armazenados, embalados e entregues de acordo com as políticas de organizações envolvidas.

A seção de agendamento das atividades de GCS estabelece a seqüencia e coordenação para as atividades identificadas de GCS e para todos os eventos que afetam a implementação do plano. O plano deve indicar a seqüencia e dependência entre todas as atividades de GCS e o relacionamento das atividades chave de GCS para os marcos (*milestones*) do projeto ou eventos.

A seção de recursos utilizados identifica os mecanismos que foram utilizados de alguma maneira para a GCS incluindo o ambiente, infra-estrutura, ferramentas de software, técnicas, equipamentos, pessoal, e treinamento necessário para implementação das atividades específicas de GCS. Deve-se documentar considerando fatores como funcionalidades, desempenho, proteção, segurança, disponibilidade, espaço necessário, equipamento, custos e restrições de tempo.

As informações sobre a manutenção do plano identificam as atividade e responsabilidades necessárias para garantir a continuidade do plano de GCS durante o ciclo de vida do projeto. O plano deve incluir o histórico de mudanças feitas no plano e declarar quem é o responsável pelo monitoramento do plano, com que freqüência as atualizações serão realizadas, como mudanças no plano devem ser avaliadas e aprovadas, e como mudanças no plano devem ser feitas e comunicadas.

O plano deve ser revisado no início de cada fase do projeto de software, alteradas em conformidade, e aprovada e distribuída para a equipe do projeto. Se o plano foi construído com procedimentos detalhados documentados em outro lugar nos apêndices ou referências, diferentes mecanismos de manutenção para estes procedimentos poderão ser mais apropriados.

Conformidade do plano de GCS com o padrão IEEE 828 - 2005

Para confirmação da conformidade de um plano de GCS com a norma IEEE 828 (2005), este mesmo plano deve satisfazer os critérios a seguir:

1. Informação mínima: O plano deve incluir as seis classes de informação de GCS especificadas anteriormente e dentro de cada classe, todas as informações exigidas declaradas devem estar documentadas dentro do plano. Se uma informação exigida e certamente não é aplicável, suas razões devem ser declaradas.
2. Formato da apresentação: Um documento, título da seção, ou até referência deve existir e que seja especificamente rotulado como "Plano de gerência de configuração de software". Dentro deste documento, cada uma das seis classes de informação deverá estar incluída. Deve-se fornecer todas as informações do plano e referências num único documento.
3. Critério de consistência: A informação documentada deve satisfazer os critérios de consistências sendo todas as atividades definidas no plano atribuído a uma unidade organizacional.
4. Declaração de conformidade: Se os critérios anteriores forem atendidos, então a conformidade de qualquer plano de GCS com a norma do IEEE Std 828-2005 deve ser declarada: "Este plano de GCS está em conformidade aos requisitos do padrão IEEE 828-2005".

2.5 Considerações finais

Através da análise das ferramentas de controle de mudanças e de versões abordadas, percebeu-se que os recursos básicos são fornecidos satisfatoriamente, o diferencial na escolha de uma delas dependerá principalmente do ambiente e da familiaridade com o usuário.

Um software de qualidade não é um estado eterno, isto é, existe a necessidade de manter esta qualidade, bem como o nível de maturidade dentro de uma empresa, sendo o plano de GCS uma referência para esta organização.

3 Elaboração de processos para Gerência de Configuração de Software

O resultado da implantação da GCS para um projeto está associado com a adoção de processos de software. O processo de GCS apóia os demais processos de desenvolvimento de software e está relacionado a diversas atividades realizadas durante o desenvolvimento de um projeto de software. A implantação de processos de software implica em afetar vários setores da organização e que podem acarretar uma maior lentidão do processo de desenvolvimento de software, incorporando práticas muitas vezes vistas como burocráticas. Assim, seu sucesso acaba dependendo mais de aspectos culturais do que dos técnicos, gerenciais e organizacionais. Porém, deve-se ressaltar para gestão de maior hierarquia da organização que a implantação destes se faz necessária principalmente para competitividade e avaliação por nível de maturidade das demais organizações.

Nesta seção, são abordados dois dos principais processos influentes no desenvolvimento do projeto que são a Gestão de mudanças e o Controle de versões e exemplificados através de diagramas de atividades da UML.

3.1 Gestão de Mudanças

A gestão de mudanças visa organizar as solicitações sugeridas ou requisitadas por clientes de um projeto que desejam um novo comportamento para o produto. O processo de gestão de mudanças requer uma participação direta com o usuário para que seja compreendido perfeitamente o que está sendo solicitado. Na figura 3.1 é proposto um processo para gestão de mudanças. Em seguida, são apresentadas as seções sobre a descrição do processo.

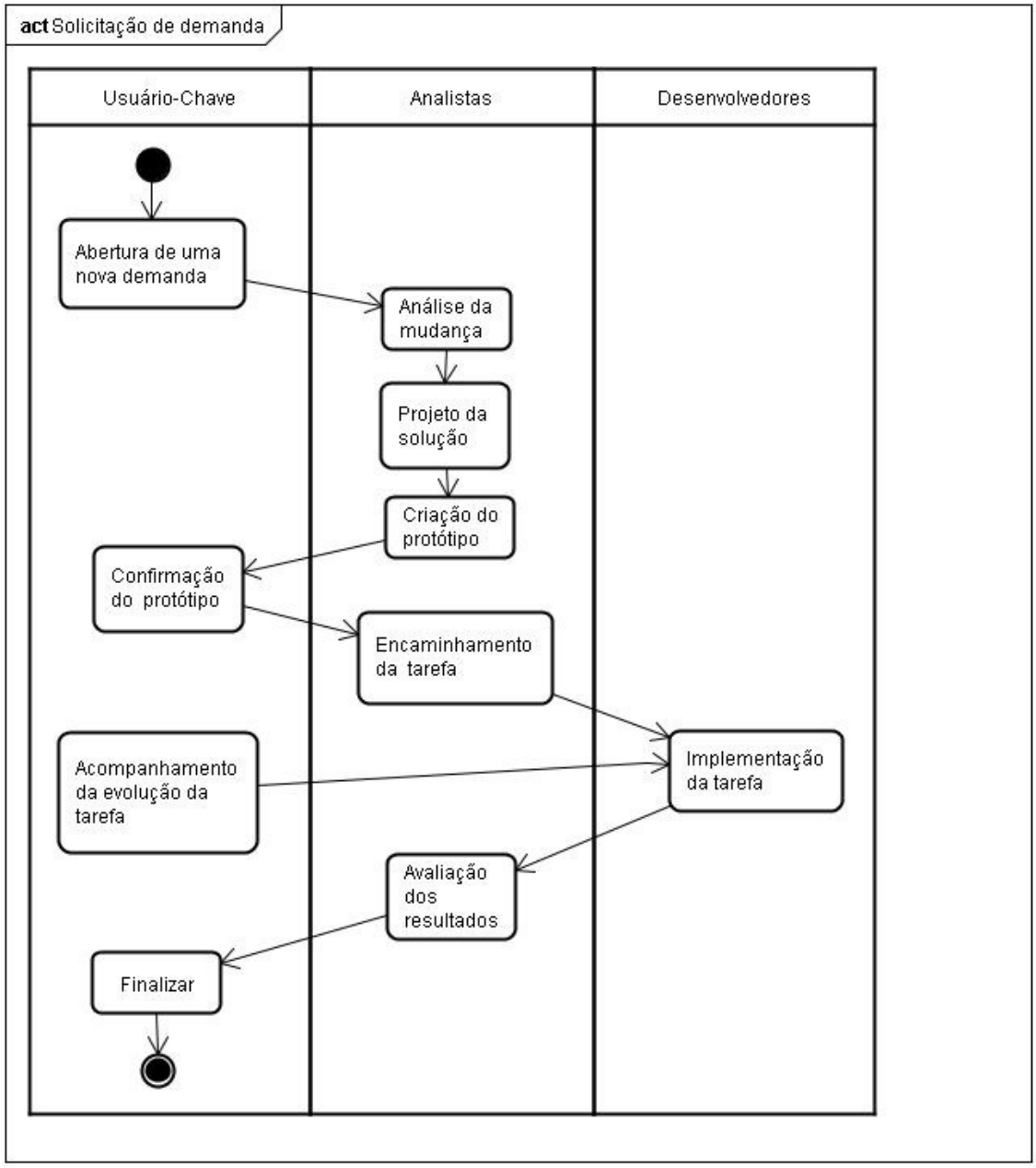

Figura 3.1 - Diagrama de atividades: Solicitação de demanda

3.1.1 Glossário

Usuário-chave: É o usuário responsável pelas solicitações de mudança em seu setor, servindo como apoio e realizando a análise das solicitações a partir de outros usuários do mesmo setor.

Tarefa: Definição formal de uma atividade a ser cumprida.

Funcionalidade: Recurso que é disponibilizado pelo sistema.

3.2 Descrição do processo de controle de mudanças

3.2.1 Abertura de uma nova mudança

O usuário-chave tem a opção de solicitar novas funcionalidades para serem implementadas na aplicação ou sistema de informação. A solicitação de uma nova funcionalidade é realizada a partir do contato pessoal do usuário com a equipe de tecnologia. A funcionalidade deverá ser refinada em um segundo momento, ou quando uma ou mais reuniões para levantamento e especificação de requisitos forem realizadas. Após o entendimento do novo requisito o usuário formaliza o pedido por meio de correio eletrônico.

A equipe de analistas responsáveis pelo projeto e manutenção do sistema se encarrega de registrar o pedido do usuário, que receberá uma notificação sobre a mudança cadastrada. A partir deste momento, o usuário-chave poderá acompanhar a execução e desenvolvimento da funcionalidade por meio do sistema de controle de mudanças. Quando uma mudança é aberta, deve ser encaminhada primeiramente para os analistas responsáveis pelo projeto, os quais farão a programação da nova demanda, como a criação de cronogramas, dimensionamento de custo ou recursos humanos. Em seguida, a demanda é encaminhada para a equipe de desenvolvedores responsável. Toda esta etapa do processo tem como objetivo e deverá estar em conformidade com padrão IEEE Std 828 (2005) que estabelece que numa solicitação de mudança devem ser registradas no mínimo as informações sobre o(s) nome(s) e as respectivas versões dos IC que serão atingidos pela mudança, o nome e organização do solicitante, data da solicitação, indicação da urgência, necessidade para atendimento da mudança e descrição da mesma.

3.2.2 Análise da mudança

Quando uma mudança é aberta, os analistas planejam sua execução, estabelecendo cronogramas, atividades, custos e recursos humanos envolvidos. Além disso, estabelecem a prioridade e complexidade da mudança aberta. Segundo o padrão IEEE Std 828 (2005) a análise necessária para determinar o impacto da mudança proposta e procedimentos de revisão dos resultados da análise devem ser especificados no plano de GCS sendo as mudanças avaliadas de acordo com seus efeitos sobre a entrega e impacto sobre os recursos do projeto.

3.2.3 Projeto da solução

Após o registro e análise da mudança, os analistas atuam na elaboração e projeto da solução, criando ou alterando artefatos produzidos durante o projeto, como, por exemplo, diagrama de classes, diagrama de sequência, casos de uso e modelo de dados.

3.2.4 Criação do protótipo

A criação de um protótipo tem como objetivo demonstrar a navegação entre telas e o subconjunto de funcionalidades da nova função para o usuário, permitindo a criação de um modelo sem as funcionalidades codificadas. O protótipo então é apresentado ao usuário-chave e fica sob avaliação do mesmo, devendo ser registrado como um IC para manter a integridade da GCS.

3.2.5 Confirmação do protótipo

A confirmação do protótipo pelo usuário-chave ocorre quando o protótipo atende os objetivos da solicitação inicial. Podendo, sugerir novos recursos que aparecerão durante a evolução da definição da tarefa solicitada.

3.2.6 Encaminhamento da tarefa

Acordadas ambas as partes, usuário-chave e analistas, as tarefas que compõem a nova funcionalidade serão encaminhadas para os desenvolvedores trabalharão no desenvolvimento das mesmas, contando sempre com o auxílio dos analistas. O encaminhamento da tarefa é feito dos analistas para os desenvolvedores responsáveis pelo projeto através da ferramenta usada para o registro de mudanças, sendo importante o registro de uma nova solicitação de mudança ser feito de acordo com o estabelecido no plano de GCS que visa atender ao padrão IEEE Std 828.

3.2.7 Implementação da mudança

De acordo com o padrão IEEE Std 828 (2005) o plano de GCS deve especificar as atividades para verificação e implementação de uma mudança aprovada, no modelo exemplificado os desenvolvedores realizarão as tarefas que forem solicitadas e tudo que for desenvolvido durante este processo deverá ser registrado através da ferramenta de controle de mudanças, permitindo o acompanhamento pelo número de solicitação gerado.

As alterações realizadas para cada solicitação aberta devem ser registradas em um *log* de ações.

3.2.8 Acompanhamento da evolução da solicitação

O usuário-chave poderá acompanhar o andamento da solicitação por meio do registro de *log* de ações existente na ferramenta de controle de mudanças.

3.2.9 Avaliação dos resultados

Assim que a mudança é concluída pelos desenvolvedores, o resultado é apresentado aos analistas para realização de testes para aprovação das alterações solicitadas inicialmente.

3.2.10 Finalização da mudança

A finalização da mudança e liberação da nova funcionalidade no sistema ocorre quando os analistas confirmam a correspondência da tarefa implementada com os requisitos especificados pelo solicitante. Uma notificação de e-mail ao usuário-chave pode ser enviada para informar sobre o encerramento da mudança. Com a conclusão e aprovação da mudança, de acordo com padrão IEEE Std 828 (2005), o plano de GCS deve registrar após uma mudança concluída no mínimo as informações sobre as mudanças associadas com as mudanças solicitadas, os nomes e versões dos itens afetados, data de verificação e parte responsável, data da versão *release* ou da instalação e a parte responsável, e o identificador da nova versão.

3.3 Controle de Versão

O controle de versão é quem ramifica e unifica a *baseline* de um projeto. Um processo bem definido para o controle de versão e com apoio de ferramentas automatizadas permite principalmente o desenvolvimento paralelo dentro de uma equipe, visando também o controle dos artefatos no repositório. É durante o processo de controle de versão é que deve ser definido entre analistas e desenvolvedores quais serão as estratégias de ramificação necessárias para melhor atender o objetivo e manter a qualidade no desenvolvimento do projeto.

A figura 3.2 apresenta um *template* para o processo de controle de versão.

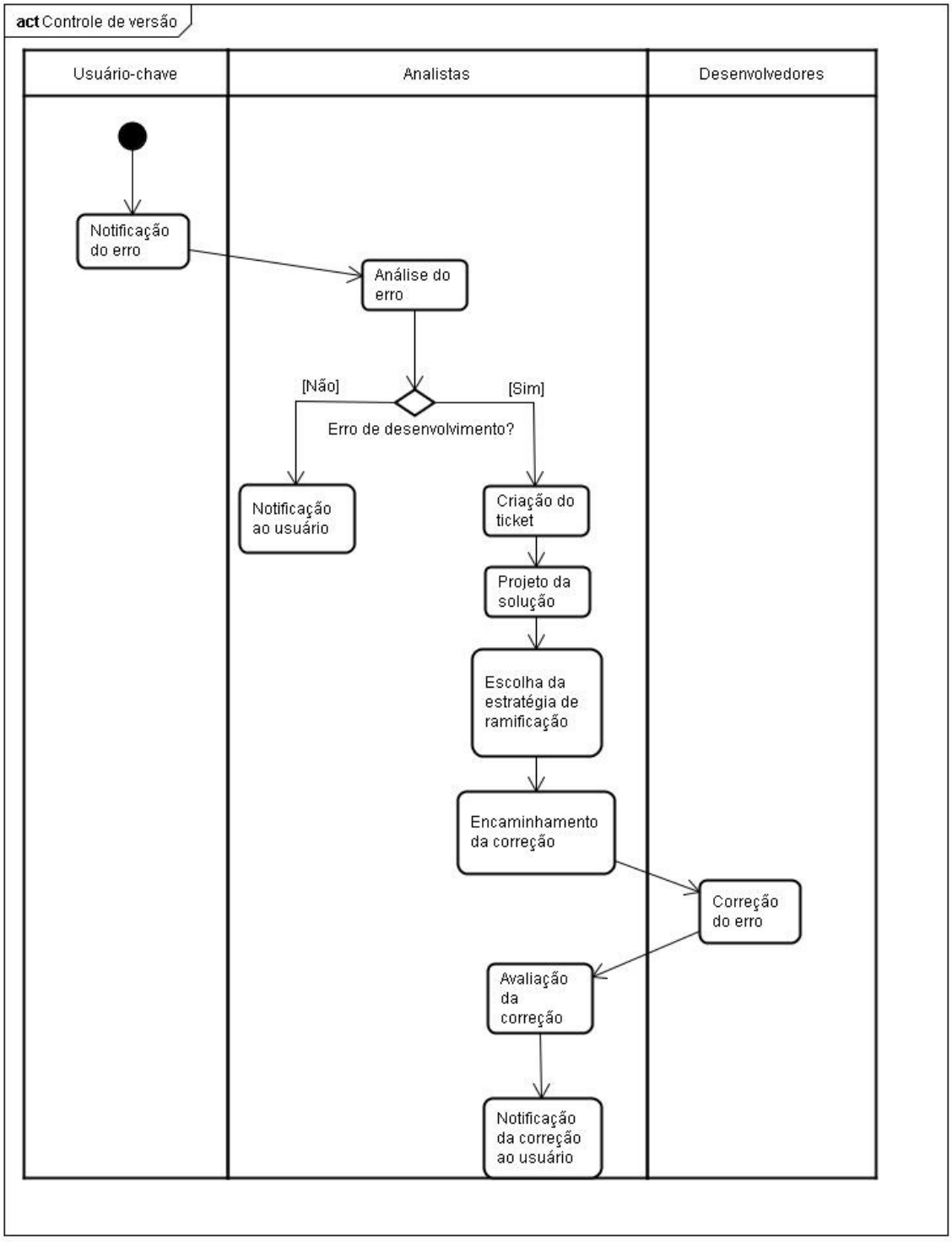

Figura 3.2 - Diagrama de atividades: Processo de controle de versões

3.3.1 Glossário

Erro: Qualquer discrepância entre os valores corretos e os resultantes de uma operação ou processamento. Podendo ser do tipo sintático ou lógico (SAWAYA, 1999).

Módulo: Divisões de componentes separados nomeados e endereçáveis que são integrados para atender aos requisitos ditados pelo problema (PRESSMAN, 2006).

3.4 Descrição do processo de controle de versão

3.4.1 Notificação do erro

Qualquer usuário que tenha encontrado um erro durante a interação com o sistema deverá notificar aos analistas para que eles tomem conhecimento de qual foi o erro encontrado durante o uso do sistema. O contato entre o usuário-chave e os analistas pode se realizar através de uma reunião, correio eletrônico ou por telefone relatando o erro.

3.4.2 Análise do erro

Após o reconhecimento do erro encontrado no sistema que foi informado pelo usuário, os analistas identificam o comportamento do mesmo para determinar sua natureza, podendo ser um erro sintático do mal uso da tecnologia ou linguagem de programação ou ainda um erro da regra de negócio. Com a confirmação e identificação do tipo do erro, cria-se um ticket do tipo “Defeito” no sistema de controle de mudanças e semelhante ao processo de solicitação de mudança que propõe atender o padrão IEEE Std 828 (2005), o ticket deve conter as seguintes informações nome(s) e as respectivas versões dos IC que serão atingidos pela mudança necessária para correção, o nome e organização do usuário que interagiu com o erro, data da notificação do erro, indicação da urgência de correção, necessidade para atendimento da mudança e descrição do erro.

3.4.3 Notificação ao usuário

De acordo com a prioridade e impacto do erro no sistema, é necessária uma notificação ao usuário para que ele tenha conhecimento sobre as ações que estão sendo decididas para correção. Esta notificação é feita após a análise do erro pelos analistas, podendo ser informada por correio eletrônico ou uma reunião.

3.4.4 Projeto da solução

Erros que precisam de uma solução imediata devido ao seu impacto ou prioridade terão uma proposta de solução rápida pelos analistas. Os erros que não possuam uma urgência ou que necessitem uma solução consistente serão encaminhados para os desenvolvedores ficando disponíveis no sistema de controle de mudanças.

3.4.5 Escolha da estratégia de ramificação

Nesta etapa do processo deve ser analisado qual será a estratégia de ramificação que melhor atende a necessidade da alteração solicitada no projeto. Como no exemplo ilustrado através do diagrama de atividades a solicitação é para uma possível correção de erro, uma das estratégias de ramificação que provavelmente melhor se adapte a ser usada está entre as estratégias de manutenção caótica, em série ou em casaca, não sendo restrita a escolha apenas entre estas estratégias.

3.4.6 Encaminhamento da correção

Com a identificação da origem erro e suas características, os analistas buscam um desenvolvedor com disponibilidade para correção. Os critérios de seleção do desenvolvedor para encaminhamento da solução levam em consideração o nível de conhecimento técnico para seleção e a familiaridade com o módulo.

3.4.7 Correção do erro

Quando é confirmado o erro de desenvolvimento, o desenvolvedor encarregado da correção deverá corrigi-lo, isto é, implementar o comportamento correto esperado pelo usuário, ele ainda pode contar com auxílio do sistema de controle de mudanças para coletar informações sobre o módulo afetado pelo erro. Assim que o desenvolvedor corrigir o erro, o módulo deve ser apresentado aos analistas para avaliação.

3.4.8 Avaliação da correção

Apresentado uma proposta de correção para defeito de um módulo pelos desenvolvedores, os analistas irão realizar testes para verificação e validação das alterações feitas no código-fonte.

3.4.9 Notificação da correção ao usuário

Após a confirmação da correção do erro pelos analistas é enviada uma notificação ao usuário-chave que fez a solicitação. Sendo o envio e a notificação feita pelos analistas.

3.5 Considerações finais

Os processos ilustrados neste capítulo são processos básicos dentro de um ambiente de desenvolvimento em que foram elaborados para atender ao padrão IEEE Std

828 (IEEE, 2005). Neste capítulo também está sintetizado os principais conceitos apresentados nos capítulos anteriores.

4 Proposta de Implantação de processos para Gerência de Configuração de Software aplicada ao ambiente de desenvolvimento do iNtegra

O objetivo deste capítulo é apresentar uma proposta de implantação dos processos apresentados no capítulo 3 no ambiente iNtegra localizado no Instituto de Ciências Exatas (ICE) na Universidade Federal de Juiz de Fora (UFJF). O projeto iNtegra tem como objetivo a criação de uma plataforma Web para abstrair a complexidade da união de dados providos de diferentes sistemas (Sistema Integrado de Gestão Acadêmica – SIGA e Google Apps) através da criação de um sistema com interfaces intuitivas e simplificadas para gerenciamento de recursos. Por meio destas interfaces é possível fornecer uma conta de e-mail institucional para cada aluno, professor e funcionário, criar listas de e-mail e calendários gerenciáveis para cada disciplina, departamento e grupo de estudo, criar diretórios on-line para compartilhamento de documentos, compartilhamento de vídeos e imagens entre usuários. O uso desta integração favorece a padronização de recursos, baixo custo relativo (recursos mantidos nos servidores), consistência de dados com o banco de dados do SIGA, maior possibilidade de colaboração entre professores e alunos e propor um sistema de boa usabilidade.

Este projeto visa atender usuários do ambiente acadêmico da UFJF tendo como principais usuários professores, técnico-administrativos e demais alunos. Atualmente, O ambiente de desenvolvimento é composto por três analistas de sistemas e alunos bolsistas da UFJF para a equipe de desenvolvimento do sistema. A equipe de bolsistas possui cinco maquinas para trabalharem e terem acesso a um repositório de testes que não é a versão *on-line* do sistema. A versão *on-line* é de acesso restrito aos analistas.

As dificuldades existentes no iNtegra que evidenciam a necessidade de um bom gerenciamento do projeto são a alta rotatividade de alunos bolsistas na equipe que impede de manter uma equipe por um prazo longo e o número crescente de demandas em curto prazo. Para auxiliar nesta gerência do projeto é necessário a escolha de ferramentas de controle de mudanças e de versão que apóiem na implantação dos processos propostos. Todos os sistemas de controle de mudanças abordadas no capítulo 2 são *open source,* isto é, possuem código aberto e oferecem notificações por e-mail além da customização do *workflow*. No ambiente de desenvolvimento do iNtegra é necessário um

SCM com estas características mas que tenha como diferenciais uma boa interface web junto com uma boa usabilidade para usuários inexperientes. O SCM que melhor se adapta ao ambiente de desenvolvimento do iNtegra é o Subversion que disponibiliza diversos *plug-ins*, possui uma versão cliente de boa interface e ainda têm uma fácil integração com o Trac.

Contando com o auxílio destas ferramentas, a implantação de processos para GCS deve ser feito com uma especificação formal e detalhada, e incluir elementos visuais de fácil compreensão para os demais usuários do sistema devido a diversidade de usuários que interagem num ambiente público como a UFJF. Recursos visuais com instruções para procedimentos de ações que tendem a reduzir a falta de referência e a redundância de perguntas. A organização no ambiente de desenvolvimento depende também da definição de um planejamento de GCS baseado no padrão IEEE Std 828 (IEEE, 2005) a serem seguidos pela equipe que especifique as responsabilidades, as atividades, cronogramas, recursos e a manutenção do mesmo como esclarece o capítulo 2.

Além do plano de GCS para auxiliar na organização do ambiente, a implantação dos processos ilustrados no capítulo 3 conta com o apoio das ferramentas para controle de mudanças e controle de versão. O Trac apóia na especificação, gerenciamento e notificações das solicitações assim como também na delegação de tarefas e acompanhamento do progresso das atividades do projeto. Conforme o processo ilustrado no capítulo 3.1 para solicitação de demandas (ou gestão de mudanças), as informações pertinentes para serem registradas no Trac são o(s) nome(s) e as respectivas versões dos IC que serão atingidos pela mudança, o nome e o departamento do solicitante, a data da solicitação, a indicação da urgência, a necessidade para atendimento da mudança e a descrição da mesma. O Subversion auxilia nas estratégias de ramificação e na integridade do projeto.

4.1 Processo de solicitação de demandas no iNtegra

A proposta de solicitação de demandas visa organizar as solicitações e o atendimento das mesmas, feitas pelos usuários do sistema iNtegra que são os professores, técnico-administrativos e demais alunos. Dentro desta proposta é estabelecido um cronograma (Tabela 4.1) de ciclo mensal que divide as etapas do processo modelo ilustrado no capítulo 3 para melhor distribuição do tempo gasto nas atividades. Este

cronograma foi criado na tentativa de evitar a sobrecarga de demandas, reduzir o esforço desnecessário da equipe de desenvolvimento e melhorar o atendimento às solicitações.

A abertura de uma nova demanda conta com o apoio da ferramenta Trac para armazenamento de cada solicitação feita pelos usuários seja através de um contato por e-mail, por telefone ou por uma reunião no ambiente de desenvolvimento. Por exemplo, um professor (usuário do sistema) que tenha a idéia de um novo recurso para o sistema de caráter comum aos demais, entra em contato com a equipe do iNtegra para solicitar esta demanda. Durante esta etapa, no momento de criação de um *ticket* (notificação do Trac que contém as informações sobre a solicitação) devem ser registradas todas as informações especificadas no capítulo 3 que atendem o padrão IEEE Std 828 (IEEE, 2005). O atendimento a uma demanda varia de acordo com o tipo de manutenção, sendo que as corretivas e adaptativas podem necessitar de um atendimento urgente enquadrando no atendimento emergencial, citado no cronograma, e as demandas de manutenções perfectivas ou preventivas, dependendo da prioridade, permitem um atendimento em longo prazo.

A etapa de análise da mudança é aonde a equipe de analistas analisam a viabilidade da solicitação, o objetivo de fornecer o recurso através do sistema, quais serão os usuários afetados e que necessitam de tal solicitação. Nesta etapa deve ocorrer uma intensa comunicação com o solicitante para realizar a especificação dos requisitos e gerar documentos que seja claro e preciso para compreensão da equipe de desenvolvimento que o terão como referência para desenvolvimento da solução.

Na etapa de projeto da solução, a equipe de analistas modela, de acordo com a especificação dos requisitos, através de Diagramas de Entidade Relacionamento (DER) as alterações necessárias no banco de dados para atender a solicitação. A partir do projeto da solução, os analistas criam um protótipo (Etapa de criação do protótipo) através de ferramentas do tipo *Website Wireframe* que demonstre a navegação entre as telas propostas para solução da solicitação. Após a criação do protótipo, este deve ser apresentado ao solicitante (usuário do sistema) que irá avaliar se o mesmo atende a suas necessidades iniciais que foi solicitada.

Assim que ocorrer a confirmação do protótipo, a equipe de analistas deve realizar uma análise para priorização de atendimento do atendimento daquela demanda, bem como ordenar qual será a ordem de atendimento as solicitações existentes para então, realizar o encaminhamento da tarefa a equipe de desenvolvedores, esta fica sendo

esclarecida como a etapa de “Encaminhamento da tarefa”, ilustrada no processo modelo do capítulo 3.

Uma das etapas deste processo que possivelmente exigirá maior esforço é a etapa de implementação da tarefa (ou desenvolvimento da solução) devido a equipe de desenvolvimento ser alunos bolsistas da UFJF que encontrarão dificuldades para desenvolver a solução sem o acompanhamento dos analistas, por isto, é necessário disponibilizar os artefatos gerados nas etapas anteriores para a equipe de desenvolvimento através do Trac. Durante esta etapa, é importante que os bolsistas mantenham o *ticket* da tarefa que está sendo desenvolvida atualizado freqüentemente, postando relatórios que informem principalmente o progresso e as dificuldades encontradas para permitir que o restante da equipe do iNtegra tenham acesso a um acompanhamento atual do andamento do solicitação. A atualização do *ticket* também permite que os usuários envolvidos na solicitação (*stakeholders*) tenham acesso ao desenvolvimento da solicitação quando o encarregado da tarefa não se encontrar no ambiente do iNtegra.

Com a proposta de solução desenvolvida pelos bolsistas, o ticket referente a solicitação, por exemplo, será atualizado para o status “Teste” para que o processo encaminhe para etapa de avaliação dos resultados, aonde serão realizados os testes necessários para validação pela equipe de analistas.

De acordo com a satisfação dos resultados atendendo o que foi solicitado, será encaminhada uma notificação ao usuário solicitante devendo o ticket da demanda ser atualizado para o status de completado no Trac.

Calendário mensal de atividades	Responsável	1º Semana					2º Semana					3º Semana					4º Semana				
		2ª	3ª	4ª	5ª	6ª	2ª	3ª	4ª	5ª	6ª	2ª	3ª	4ª	5ª	6ª	2ª	3ª	4ª	5ª	6ª
Atividade 1	Usuários																				
Atividade 2	Analistas																				
Atividade 3	Analistas																				
Atividade 4	Analistas e Usuários																				
Atividade 5	Analistas																				
Atividade 6	Analistas e Usuários																				
Atividade 7	Analistas e Desenv.																				
Atividade 8	Analistas e Desenv.																				

Tabela 4.1 - Cronograma de atividades

Atividade	Descrição da atividade
1	Abertura de solicitação
2	Análise da solicitação, projeto da solução e criação do protótipo
3	Avaliação, priorização de demandas e encaminhamento da tarefa
4	Especificação de requisitos
5	Planejamento
6	Avaliação dos resultados
7	Implementação e testes
8	Atendimento emergencial

Tabela 4.2 - Descrição das atividades

4.2 Processo de controle de versão no iNtegra

A proposta para o processo de controle de versão inicia-se quando um usuário do sistema se deparar com um erro, por exemplo, a secretária do ICE não conseguir reservar um recurso através do sistema. O usuário então deve entrar em contato com a equipe do iNtegra seja por reunião, e-mail ou telefone com a notificação do erro apresentado. É recomendado este contato ser realizado preferencialmente por reunião ou telefone devido a dificuldade do usuário em expressar o erro apresentado em uma linguagem técnica. Para ajudar no contato realizado por e-mail pode-se disponibilizar seja através do iNtegra *on-line* ou através da secretaria, um questionário modelo para identificação do comportamento do erro a ser respondido através do contato por e-mail.

Com a ciência da notificação do erro, a equipe de analistas analisa este erro para concluir se é originado pelo incorreto desenvolvimento da funcionalidade ou por desconhecimento do usuário dos procedimentos para interação com o recurso do sistema. Erros por desconhecimento do usuário devem ser respondidos o mais breve possível, mas se for uma solicitação de erro que persiste por outros usuários, percebe-se a necessidade de uma análise para uma nova demanda de manutenção perfectiva. No caso em que o erro é do sistema, a equipe de analistas cria um *ticket* no Trac, assim como ilustrado no capítulo 3, aonde a próxima etapa de projeto da solução, dependerá da indicação de urgência, tipo de erro e disponibilidade de atendimento.

Com a disponibilidade para o atendimento da notificação de erro, a elaboração do projeto de solução define, além das alterações necessárias no banco de dados, a escolha da estratégia de ramificação do projeto. A escolha de estratégia de ramificação é feita com apoio da ferramenta Subversion e depende de vários fatores, como exemplo, os bolsistas responsáveis pelo desenvolvimento nem sempre estão nivelados em quesitos de experiência e técnica. Por isto, dependendo do tipo de demanda e o prazo para entrega, é

comum cada desenvolvedor atender uma solicitação, sendo escolhida a estratégia de ramificação de organização por desenvolvedores, onde cada bolsista ramifica uma versão do projeto do repositório e desenvolve a solução. Quando ocorre o caso onde o responsável pelo último *commit* (que incluiu erros na versão do repositório de desenvolvimento) não se encontra, a estratégia de manutenção em cascata é mais adequada, isto é, restaura-se a última versão do servidor de desenvolvimento que não possui erros num ramo auxiliar (para a máquina do bolsista) e desenvolve-se esperando até que a versão no ramo principal seja corrigida.

Realizado a escolha da estratégia de ramificação, os analistas encaminham a solução preferencialmente para o bolsista desenvolvedor encarregado pela implementação daquele módulo. Se este desenvolvedor está comprometido a outras tarefas de alta prioridade, ou quaisquer outros motivos que o impedem de desenvolver a solução naquele momento e que existe a possibilidade de outro bolsista corrigir o erro, a tarefa deverá ser encaminhada para o segundo bolsista.

A correção do erro realizada pelo bolsista desenvolvedor contará com o apoio da documentação desenvolvida naquele módulo e das informações disponíveis na notificação do erro através do Trac.

A etapa de avaliação da correção realizará pela equipe de analistas a verificação e validação da correção implementada pela equipe de desenvolvimento, com o resultado satisfatório dos testes, a versão *on-line* é então atualizada e uma notificação informando sobre a correção deve ser enviada ao usuário.

4.3 Considerações finais

De uma maneira que não fique restrita a empecilhos, por exemplo, número de integrantes da equipe do iNtegra, este capítulo teve como objetivo propor como seria a implantação dos processos modelados no capítulo 3 dentro do ambiente iNtegra.

5 Considerações Finais

Este trabalho permitiu um estudo sobre conceitos de engenharia de software com foco em aprimorar a qualidade de processos, modelos de maturidade, comparação entre ferramentas de controle de versão e a implantação de processos. O tempo de vida relativamente pequeno do projeto iNtegra e a pouca documentação sobre o mesmo inviabilizou um estudo de caso mais profundo. A proposta apresentada neste trabalho visa fornecer na forma de documento uma especificação formal para implantação de dois processos fundamentais da engenharia de software no desenvolvimento do projeto. A proposta de implantação de processos de GCS para o iNtegra, ilustrada no capítulo 4, se restringiu a não ser tão específica a ponto de limitar o número de desenvolvedores ou integrantes da equipe, tendo com objetivo uma estrutura a ser seguida baseada na experiência a qual estive durante meu período de participação no projeto.

Uma das prováveis dificuldades a ser encontrada durante a implantação destes processos é uma negação por parte dos usuários como sendo apenas, a partir do ponto de vista do usuário, mais um processo burocrático ineficiente dentro de um ambiente público, porém deve ser contornado com a demonstração dos resultados.

Os resultados esperados com a implantação são a melhora na qualidade dos processos e do sistema, melhora na produtividade (especificamente no atendimento as demandas) e oferecer a experiência com conceitos da engenharia de software aos bolsistas. Como trabalho futuro pode ser feito a análise do resultado da implantação destes processos realizando comparações levando em consideração principalmente a satisfação na equipe do projeto, a produtividade e a qualidade do processo e do produto. Conforme aparecerem resultados satisfatórios na implantação dos processos, pode ser realizada também a documentação de um plano de GCS do projeto para permitir a evolução baseando-se nos principais modelos de maturidade.

Referências bibliográficas

APPLETON, Brad; BEREZUK, Stphen P.; CABRERA, Raplh; ORENSTEIN, Robert. **Streamed Lines: Branching Patterns for Parallel Software Development**.1998.

CHRISSIS, Mary Beth; KONRAD, Mike; SHRUM, Sandy. **CMMI: Guidelines for Process Integration and Product Improvement**. Boston: Addison-Wesley, 2002.

COLLINS-SUSSMAN, Ben; FITZPATRICK, Brian W; PILATO, C. Michael. **Version Control with Subversion**. 2006. Disponível em <http://svnbook.red-bean.com/en/1.1/index.html> Acesso em: 24 abr. 2011.

DANTAS, Cristine. **Gerência de configuração de software**. Engenharia de Software Magazine, Rio de Janeiro, ano 1, ed.2, p.16-24, DevMedia 2008.

DAFLON, Leandro. **Gerência de Configuração CMMI**. In: Seminário CMMI, 2005, Rio de Janeiro. Disponível em: <http://wiki.les.inf.puc-rio.br:8080/gqs/space/PDFs+Gerencia+Configuracao/09+1+Palestra+PAGerConfiguracaoCMMI+02052005.pdf>. Acesso em: 25 set. 2010.

DART, Susan. **Concepts in configuration management systems**, Proceedings of the 3rd international workshop on Software configuration management, p. 1-18, June 12-14, 1991.

EDGEWALL SOFTWARE, 2007, **The Trac Project**. Disponível em <http://trac.edgewall.org> Acesso em: 24 abr. 2011.

EDGEWALL SOFTWARE, 2007, **Trac Instalattion Guide**. Disponível em <http://trac.edgewall.org/wiki/TracInstall> Acesso em: 24 abr. 2011.

EDGEWALL SOFTWARE, 2007, **Customizing the Trac Interface**. Disponível em <http://trac.edgewall.org/wiki/TracInterfaceCustomization> Acesso em: 24 abr. 2011.

EDGEWALL SOFTWARE, 2007, **Email Notification of Ticket Changes**. Disponível em <http://trac.edgewall.org/wiki/TracNotification> Acesso em: 24 abr. 2011.

EDGEWALL SOFTWARE, 2007, **The Trac Ticket Workflow System**. Disponível em <http://trac.edgewall.org/wiki/ TracWorkflow> Acesso em: 24 abr. 2011.

FRANCO, Alexandre Luís. **Processo de Gerencia de Configuração.** [2010?] Disponível em <http://sites.google.com/site/alexandreluisfranco2/ISO12207-7.GerncíadeConfigurao.pdf> Acesso em: 07 Nov. 2010.

GIT, 2011, **GIT User's Manual**. Disponível em <http://www.kernel.org/pub/software/scm/git/docs/user-manual.html> Acesso em: 24. abr. 2011.

GIT , 2011, **GIT Wiki Homepage**. Disponível em <https://git.wiki.kernel.org/> Acesso em: 24. abr. 2011.

GREAVES, David. **GIT Manual Page**. GIT. Disponível em <http://www.kernel.org/pub/software/scm/git/docs/v1.7.4.5/git.html> Acesso em: 24. abr. 2011.

IEEE, 1996, Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineers.

IEEE, 2005, Std 828 - IEEE Standard for Software Configuration Management Plans, Institute of Electrical and Electronics Engineers

KOSCIANSKI, André; SOARES, Michel Dos Santos. **Qualidade de software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. São Paulo: Novatec, 2007.

MANTIS. **Features**. Disponível em <http://www.mantisbt.org/wiki/doku.php/mantisbt:features> Acesso em: 24 abr. 2011.

MANTIS. **Mantis Bug Tracker.** MantisBT Group, 2009. Disponível em <http://www.mantisbt.org/> Acesso em: 08 dez. 2011.

MANTIS. **Mantis Bug Tracker Administration Guide**. Disponível em <http://docs.mantisbt.org/master/en/administration_guide.html> Acesso em: 24 abr. 2011.

MANTIS. **The MantisBT Manual.** Disponível em <http://www.mantisbt.org/manual/> Acesso em: 24 abr. 2011.

MURTA, Leonardo Gresta Paulino. **Gerência de Configuração: Ramificação e Integração**. Disponível em: <http://www.ic.uff.br/~leomurta/courses/2011.1/gc/aula5.pdf> Acesso em 24 mai. 2011

O'SULLIVAN, Bryan. **Mercurial: The Definition Guide**. O'Reilly Media, 2009. Disponível em: <http://hgbook.red-bean.com/read/> Acesso em: 24 abr. 2011.

PRESSMAN, Roger S. **Engenharia de software.** 6.ed. São Paulo: McGraw-Hill, 2006

SAWAYA, Márcia Regina. **Dicionário de Informática e Internet**. 3 ed. São Paulo: Nobel, 1999.

SEI, 2006, CMMI® *for Development*, *Version* 1.2, Carnegie Mellon University.

SHAHRI, Hamid Haidarian; HENDLER, James A.; PORTER, Adam A. **Software Configuration Management Using Ontologies.** Proceedings of the 3rd International Workshop on Semantic Web Enabled Software Engineering at the 4th European Semantic Web Conference (ESWC'07), Innsbruck, Austria, June 6-7, 2007.

SOFTEX MPS.BR - Guia de Implementação - Parte 2: Nível F (Versão 1.1), Associação para Promoção da Excelência do Software Brasileiro, 2007.

SOFTEX (2009) Modelo MPS – Melhoria de Processo do Software Brasileiro, Guia Geral do MPS, Guia de Aquisição do MPS, Guia de Implementação do MPS, e Guia de Avaliação MPS. Sociedade Softex, Brasil. Disponível em http://www.softex.br/mpsbr , acessado em 15/06/2011.

SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Pearson Addison-Wesley, 2007.

THE BUGZILLA TEAM. **Features**. Disponível em <http://www.bugzilla.org/features/> Acesso em: 24. abr. 2011.

THE BUGZILLA TEAM. **Installation**. Disponível em: <http://www.bugzilla.org/docs/4.0/en/html/installation.html>. Acesso em: 24 abr. 2011.

THE BUGZILLA TEAM. **Lyfe Cycle of a Bug**. Disponível em: <http://www.bugzilla.org/docs/4.0/en/html/lifecycle.html>. Acesso em: 24 abr. 2011.

THE BUGZILLA TEAM. **OS-Specific Installation Notes**. Disponível em: <http://www.bugzilla.org/docs/4.0/en/html/os-specific.html>. Acesso em: 24 abr. 2011.

THE BUGZILLA TEAM. **The Bugzilla Guide:** 4.0.2 Release. Disponível em: <http://www.bugzilla.org/docs/4.0/en/html/>. Acesso em: 24 abr. 2011.

WARE LAB. **The Mantis Issue Workflow.** Disponível em: <http://www.warelab.org/blog/?p=24>. Acesso em: 11 out. 2011