Aplicando Padrões de Gerência de Configuração de Software em Projetos Geograficamente Distribuídos

Article · August 2005

Source: OAI

CITATIONS
2

READS
20

2 authors, including:

Lucas Cordeiro
The University of Manchester
158 PUBLICATIONS **949** CITATIONS

SEE PROFILE

Aplicando Padrões de Gerência de Configuração de Software em Projetos Geograficamente Distribuídos[1]

Dario André Louzado, Lucas Carvalho Cordeiro

Siemens Com Mobile Devices – Siemens Eletroeletrônica S.A.
Manaus – AM – Brazil

{dario.louzado,lucas.cordeiro}@siemens.com

Abstract: *Software Configuration Management (SCM) plays an important role in software development projects by controlling the consistency of artifacts during the whole project life cycle. In this article we discuss how a well-known SCM pattern language was applied in a medium size outsourced project. For each applied pattern we explain the motivations, the context, the forces considered and the positive and negative consequences. We present the ideas by looking to the SCM system as continuous evolving system as the patterns are applied. Special nuances, common to the complex world of outsourcing, are also emphasized.*

Keywords: *patterns, software configuration management, outsourcing.*

Resumo: *Gerência de Configuração de Software (SCM) desempenha um papel importante no desenvolvimento de projetos de software, controlando a consistência dos artefatos ao longo do ciclo de vida do projeto. Neste artigo é discutido como uma linguagem de padrões de SCM conhecida foi aplicada em um projeto terceirizado de tamanho médio. Para cada padrão aplicado são expostas as motivações, contexto, forças consideradas bem como as conseqüências positivas e negativas. As idéias são apresentadas olhando para o sistema de SCM como um sistema que evolui na medida em que os padrões são aplicados. Nuances especiais, comuns ao complexo mundo da terceirização, são também enfatizadas.*

Palavras-chave: *padrões, gerência de configuração de software, terceirização.*

1. Introdução

Uma disciplina da engenharia de software que vem ganhando crescente destaque em projetos de software é a *gerência de configuração do software*, ou *software configuration management* – SCM. A razão para tanto destaque é muito simples. Se entendermos todo o processo de desenvolvimento de software como um software [1], SCM pode ser vista como o subsistema de entrada-saída (*I/O*) deste software.

Indo um pouco mais além, SCM responde pelo controle transacional dos artefatos de software, isto é, pelo controle da consistência do produto de trabalho produzido pelos desenvolvedores ao longo de todo o ciclo de vida do projeto. Por exemplo, ao receber o código-fonte de diferentes desenvolvedores, o gerente de configuração do software organiza este material em um espaço de trabalho (*workspace*) checa as consistências e dispara o processo de geração de *builds*. Como resultado deste último processo, tem-se uma versão *intermediária,* ou incremento, do software em construção ou manutenção.

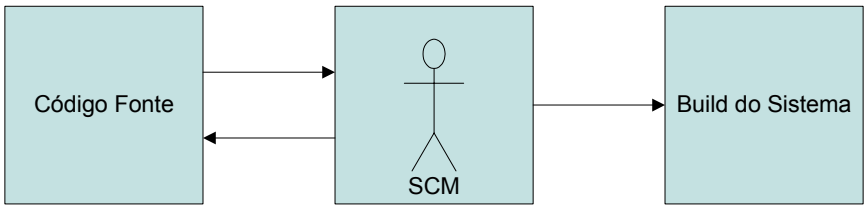

Figura 1 - Sistema de entrada e saída SCM

Conforme mostrado na figura 1, o sistema de *I/O* SCM recebe código-fonte (entrada), organiza, verifica as diferenças e consistências e produz um *build*, uma saída, portanto, deste processo. O exemplo é simples, mas serve para ilustrar o papel da gerência de configuração em projetos de software. Um *build* que apresenta problemas de compilação ou integração poderia sacrificar um ou mais dias de trabalho de uma equipe inteira de desenvolvedores ou testatores. *Builds* com este tipo de problema dificultam inclusive a gerência do projeto pelos líderes, pois a noção de progresso é consideravelmente ofuscada.

Na prática, as atividades, considerações e verificações realizadas pelo gerente da configuração são bem menos triviais que o exemplo acima. Neste artigo, estaremos navegando pela linguagem de padrões de gerência de configuração definida em [3], objetivando discutir, para cada padrão aplicado e para a linguagem como um todo, quais as considerações, dificuldades e adaptações realizadas no contexto da Siemens Communications R&D.

É importante salientar que o propósito principal deste artigo é focar em assuntos estratégicos de SCM tais como: *práticas, políticas e organização.* Com este objetivo em mente, o apêndice A fornece uma visão geral das ferramentas utilizadas no projeto. Além disso, por questões de confidencialidade, o artigo abordará apenas do uso da técnica, omitindo qualquer tipo de informação que envolva o escopo e a estratégia do projeto analisado.

2. Contexto Geral

O projeto analisado tem por objetivo produzir um software de uso *desktop* (com aproximadamente 80,000 LOC) destinado a usuários finais de celulares Siemens. Por ser destinado ao usuário final em um mercado de massa, trata-se de um projeto crítico o qual demanda um controle rigoroso na configuração do software. A Siemens Communications é uma organização com uma evidente política de presença no mercado (*time-to-market),* o que evidencia a necessidade de controle sobre o projeto, status da configuração e da qualidade do código.

Adicionalmente, o projeto é desenvolvido por quatro parceiros situados fisicamente em localidades diferentes – Figura 2. Esta realidade demanda uma boa comunicação e uma definição clara de responsabilidades, situando ainda mais a gerência da configuração do software como um instrumento chave neste processo.

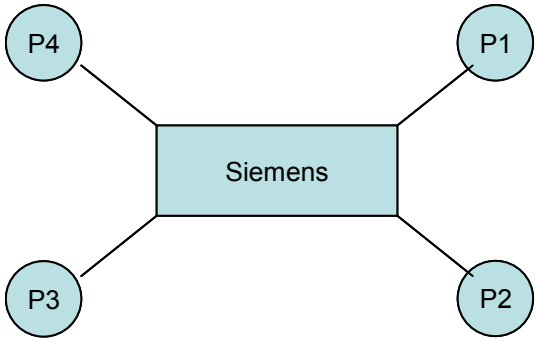

Figura 2 - Siemens e seus parceiros

Outra particularidade encontrada foi a necessidade de separar um pouco a gestão dos artefatos de software (atividades de integração e geração do *build*). A primeira é atribuição do gerente de configuração (C*onfiguration Manager*, doravante denominado CM). A segunda é atribuição do gerente de build (*Build Manager*, doravante denominado BM).

3. Organização, arquitetura e gerência de configuração de software

Organizações estruturam-se de acordo com o mercado para lançar produtos ou soluções [2]. Esta estruturação influência fortemente a arquitetura do software. Conforme os sistemas de software tornam-se mais completos, a arquitetura passa a influenciar a organização e suas decisões.

Uma influência importante é a *localização* do trabalho, isto é, como um pacote de trabalho é atribuído a uma determinada equipe. Dois componentes com muita proximidade e dependência são atribuídos de forma localizada a um time de desenvolvedores, minimizando a demanda por canais de comunicação, o que adiciona lentidão e riscos ao projeto.

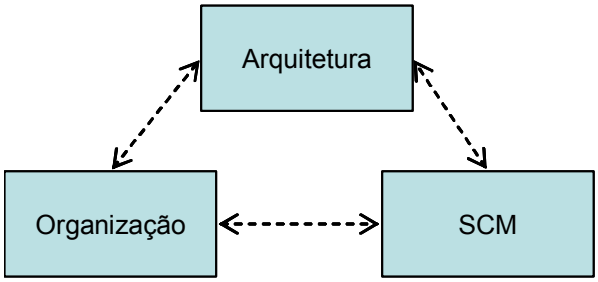

Figura 3 - Influências entre organização, arquitetura e SCM

De uma maneira geral, o padrão *Architecture Follows Organization* [2] discute esta questão e sobre como a estrutura arquitetural desenha os canais de comunicação em uma organização. Com esta abordagem em mente, o ciclo de influências proposto pela Figura 3, tanto as estruturas organizacional quanto arquitetural influenciam diretamente as práticas, políticas, planejamento e as ferramentas destinadas à gerência da configuração do software.

Estudando estas dependências conceituais, foi desenvolvida uma linguagem de padrões destinada à gerência da configuração [3]. Esta linguagem classifica os padrões em duas categorias:

- *Codeline:* padrões relacionados ao controle de versão e ao isolamento de iniciativas distintas em linhas de codificação, tipicamente implementadas em ferramentas de controle de versão com o conceito de *branches*.
- *Workspace:* padrões relacionados ao agrupamento de versões específicas de artefatos do projeto em áreas de trabalho a fim de suportar diferentes atividades do projeto. Exemplos: gerar um *build*, executar *smoke tests*, desenvolver funcionalidades, integrar software, entre outras.

A linguagem de padrões SCM vem sendo refinada há pelo menos quatro anos pelos autores originais e este artigo adiciona uma contribuição a partir de uma experiência em projeto com times geograficamente distribuídos. A Figura 4 mostra as interações entre os padrões da linguagem em estudo.

É importante observar a partir do mapa de linguagem de padrões SCM (figura 4), que a seta padrão A → padrão B significa que padrão A precisa do padrão B para completá-lo [3]. Deste modo, o padrão *Task Level Commit* deve ser implementado para que o *Integration Build* funcione.

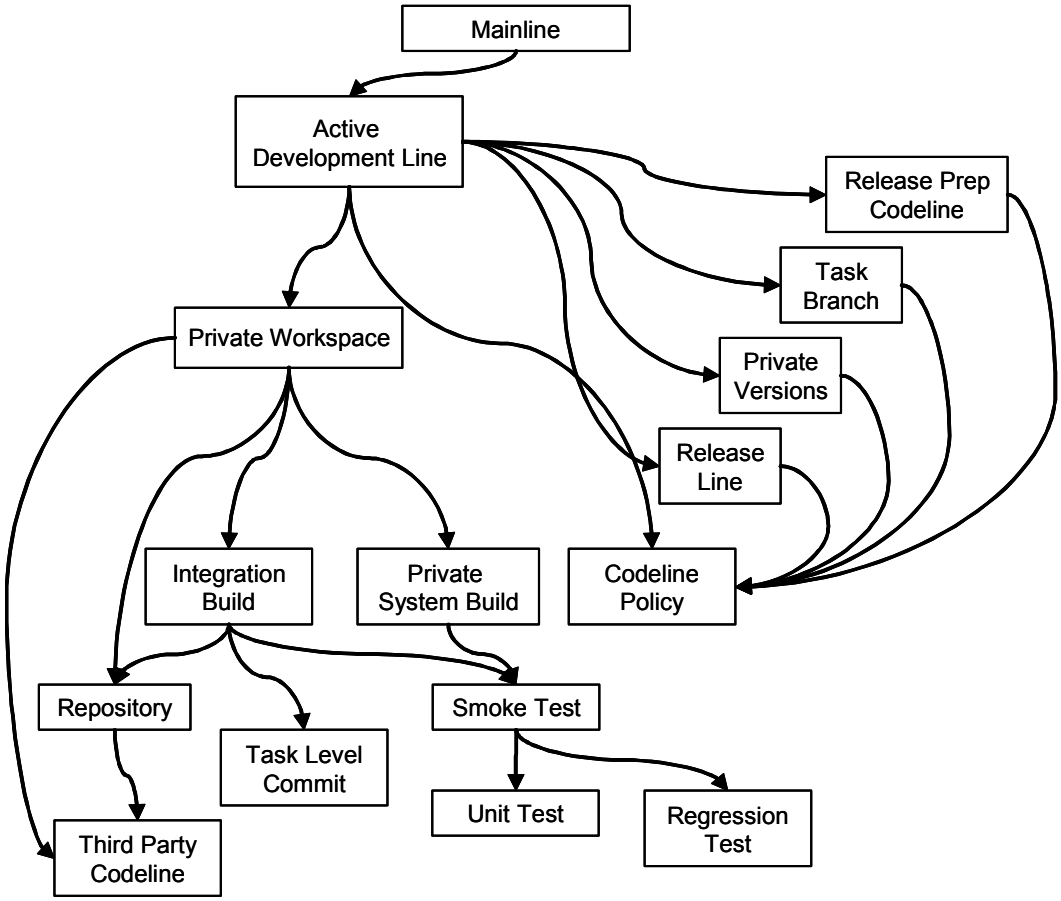

Figura 4- Padrões de Gerência de Configuração de Software [3]

4. Padrões de gerência de configuração de software

Esta seção descreve cada padrão de gerência de configuração de software utilizado no projeto. A aplicação destes padrões é determinada de acordo com as decisões de organização e arquitetura, conforme mencionado na seção 3. Deste modo, alguns padrões de SCM propostos por [3] não foram aplicados e os demais sofreram adaptações considerando o contexto específico do projeto observado.

4.1 Padrão *Mainline*

4.1.1 Contexto de Aplicação

Como mencionado na seção 2, o projeto é desenvolvido por quatro parceiros situados fisicamente em localidades diferentes. Cada parceiro possui sua estrutura organizacional e diferentes tipos de ferramentas para lidar com gerência de configuração. Sendo assim, foram criadas cinco diferentes linhas de codificação (*codelines*), uma para cada parceiro e uma principal gerenciada pela Siemens.

4.1.2 Problemas enfrentados

Alguns componentes possuem dependências entre si, por exemplo, qualquer mudança na interface e/ou comportamento do componente afeta o trabalho de um ou mais parceiros. Se todos os parceiros possuem linhas de codificação diferentes, como resolver este problema de dependência?

4.1.3 Adaptações

Para que fosse possível utilizar cinco diferentes linhas de codificação e ter maior controle de todas as mudanças de interface/comportamento dos componentes, foi desenvolvido um processo de *comunicação de mudança de interface.* Neste processo, o parceiro realiza a mudança na interface/comportamento do componente e depois notifica todos os parceiros afetados.

Esta adaptação é válida, pois, o projeto é norteado por uma arquitetura baseada em componentes e interfaces bem definidas. Ciclos de integração, com freqüência semanal, garantem que apenas um *codeline* (a linha de codificação de onde são gerados os *builds* e *releases*) conterá as versões definitivas do produto.

4.1.4 Contexto Resultante

Depois da implementação deste processo, cada parceiro foi capaz de trabalhar em sua própria linha de codificação. O processo de *comunicação de mudança de interface* posasibilitou uma melhor comunicação, de acordo com as dependências arquiteturais. Esta comunicação aprimorada permitou uma maior transferência de responsabilidade para os parceiros. O foco da Siemens pode ser mantido no controle da linha de codificação principal. A linha principal, incrementada a partir das integrações semanais, é a única referência – não-ambígua, portanto – para versões oficiais do produto.

4.2 Padrão *Active Development Line*

4.2.1 Contexto de Aplicação

O desenvolvimento ocorre em cinco linhas ativas de codificação, uma para cada parceiro, incluindo desenvolvimento interno. Isto implica dizer que, para cada uma das linhas, uma partição do sistema (em termos de sub-sistemas e componentes) é desenvolvida e entregue por cada um dos parceiros envolvidos no projeto.

4.2.2 Problemas enfrentados

A coordenação das entregas, a fim de garantir um único sistema funcionando não é uma tarefa simples. Muitos problemas de retrabalho ou esforço elevado de integração foram encontrados.

4.2.3 Adaptações

Toda entrega de código do parceiro é acompanhada por notas de liberação (*release notes)*. As notas de liberação têm o propósito de fornecer as condições atuais da entrega, ou seja, quais componentes foram adicionados ou modificados, quais bugs foram resolvidos, quais as limitações de cada componente e assim por diante. O gerente de configuração de software é responsável por revisar as notas de liberação de cada entrega e comunicar aos parceiros eventuais problemas de consistência.

4.2.4 Contexto Resultante

Com o adequado preenchimento das notas de liberação é possível integrar e gerar um novo *build* do sistema e, por conseguinte disponibilizar uma versão estável do mesmo na linha de codificação principal. Esta prática, tornou viável a implantação de uma estratégia de *integração por estágios*, na qual um mesmo ciclo de integração é quebrado em ciclos menores de acordo com as dependências entre os componentes.

4.3 Padrão *Private Workspace*

4.3.1 Contexto de Aplicação

Diferentes parceiros possuem diferentes estruturas organizacionais, cultura de trabalho e assim por diante. O isolamento geográfico demanda um certo isolamento para a construção do software.

4.3.2 Problemas enfrentados

Com o uso de uma linha de codificação por parceiro, pode-se isolar as mudanças feitas por cada um deles e obter um melhor controle do software sendo desenvolvido. Em cada uma dessas linhas, cada desenvolvedor de cada parceiro pode gerar o seu espaço de trabalho (*workspace*).

Mesmo usando diferentes linhas de codificação, pôde ser observada a utilização de forma intrusiva, isto é, parceiro P2 modifica componentes atribuídos ao parceiro P1 (conforme mostrado na Figura 5). Este problema causa desperdício de esforço visto que não há garantia de consistência nas versões produzidas por P1.

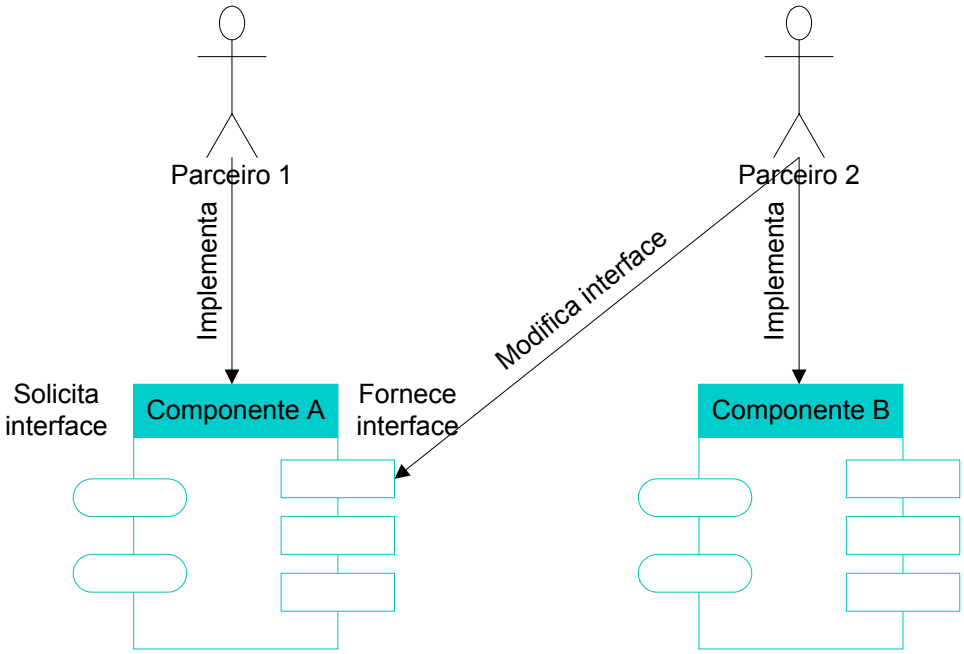

Figura 5 – Mudança de interface de componente

4.3.3 Adaptações

O estabelecimento de um processo de comunicação de mudança de interface e a separação de responsabilidades atenuou os problemas enfrentados. A arquitetura bem definida e o reforço da atribuição das responsabilidades desempenhou um papel fundamental na redução deste tipo de problema.

4.3.4 Contexto Resultante

Melhoria na comunicação entre os times e na integração dos componentes implicando em *builds* mais estáveis e entregas mais controladas.

4.4 Padrão *Integration Build*

4.4.1 Contexto de Aplicação

Uma data e horário do dia são marcados para cada parceiro realizar sua entrega. Duas entregas são realizadas na semana, de acordo com as dependências – *integração por estágios*. Esta abordagem favorece a integração gradual do produto. Cada uma das entregas deve ser acompanhada por notas de liberação e um rótulo *tag* atribuída à versão específica dos componentes sendo entregues (conforme mostrado na figura 5). As notas de liberação devidamente preenchidas e a *tag* associada aos componentes facilitam a integração e a geração de um novo *build* do sistema.

4.4.2 Problemas enfrentados

Algumas entregas não foram acompanhadas de notas de liberação devidamente preenchidas, o que dificultou a implementação da *integração por estágios*. Além disso, a *tag* era associada a todos os componentes do sistema, dificultando a assimilação do que estava sendo entregue de fato.

4.4.3 Adaptações

O gerente de configuração de software é responsável por verificar se a *tag* foi corretamente atribuída aos devidos componentes e verificar, para cada entrega de parceiro, se as notas de liberação condizem com o conjunto de componentes entregues. Com estas informações em mãos, o SCM comunica aos parceiros da inconsistência da entrega. Atualmente, um *checklist* foi desenhado para validar e fornecer feedback de cada entrega.

4.4.4 Contexto Resultante

*Build*s de integração são produzidos de forma sistemática, com periodicidade definida e controlada. No caso do projeto em questão, usou-se a freqüência semanal.

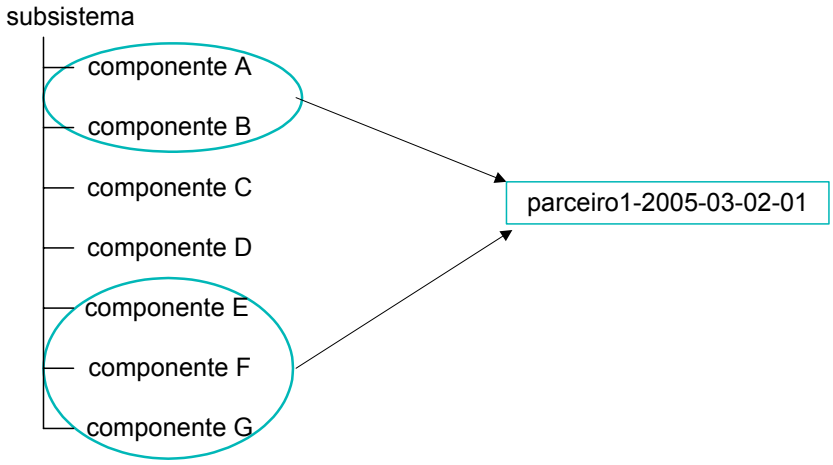

Figura 6: Parceiro 1 atribuindo uma *tag* aos componentes

4.5 Padrão *Third Party Codeline*

4.5.1 Contexto de Aplicação

Cada parceiro possui uma linha de codificação no sistema de controle de versão. O ciclo de vida da linha de codificação (atualização, desenvolvimento, teste de integração interna e assim por diante) está sob responsabilidade do parceiro. As atualizações bem

como as entregas devem ocorrer de acordo com políticas bem estabelecidas no início do projeto.

4.5.2 Problemas enfrentados

Má gestão das linhas de codificações dedicadas aos parceiros. Muita demanda interna para integrar e gerar builds, o que afeta diretamente o caminho crítico do projeto.

4.5.3 Adaptações

Papéis e responsabilidades foram definidos, enfatizados e cobrados, para cada parceiro envolvido. Neste sentido, foram desenvolvidos, documentados e divulgados todos os procedimentos e políticas de gerência de configuração.

4.5.4 Contexto Resultante

Seguindo todos os procedimentos estabelecidos pela Siemens, cada parceiro foi capaz de cuidar, com certo grau de independência, do ciclo de vida da sua linha de codificação. Boa parte do caminho crítico do projeto foi aliviada, graças a este redirecionamento de responsabilidade.

4.6 Padrão *Task Level Commit*

4.6.1 Contexto de Aplicação

Uma tarefa (ou *task*), para o projeto analisado, pode ser mapeada em uma entrega individual por parceiro. Por exemplo, implementar um novo componente ou serviço. Cada parceiro realiza a sua entrega em uma linha de codificação isolada, usando controle de versão a partir de *tag*.

4.6.2 Problemas enfrentados

Cada parceiro é responsável por tarefas e pelo ciclo de vida de sua linha de codificação individual. Em alguns momentos, quando um parceiro depende fortemente da modificação de outro, o dependente terá que aguardar pela conclusão da tarefa, o que pode significar tempo improdutivo de espera.

4.6.3 Adaptações

Isolamento das linhas de codificação por parceiro e a entrega baseada em *tag*. Para contornar o problema da espera por dependência, usa-se o conceito de *snapshot*, ou fotografia. Nesta abordagem, o parceiro causador da dependência prioriza as suas atividades e, antes da conclusão da tarefa, aplica um rótulo em sua linha de codificação tornando a versão disponível aos demais interessados. Uso intensivo de notas de liberação com o propósito de comunicar efetivamente o andamento das modificações para todos os envolvidos.

4.6.4 Contexto Resultante

Percepção consistente do andamento do projeto pelos líderes, devido ao incremento consistente de funcionalidades no software. Maior controle sobre os *builds* globais do produto em construção a partir de diferentes entregas. Necessidade de pessoas dedicadas à tarefa de monitorar o andamento das modificações e integrar componentes.

4.7 Padrão *Smoke Test*

4.7.1 Contexto de Aplicação

Mesmo entregas bem estruturadas, tal como proposto em *Task Level Commit,* demandam uma verificação mínima de consistência. Eventualmente versões são enviadas para outros times remotamente localizados ao redor do globo a fim de executar testes de diversos tipos.

4.7.2 Problemas enfrentados

Contínuo balanceamento entre *time-to-market* e estabilidade de cada *build*. Difícil decisão entre lançar um *build* antes ou depois dos *Smoke Tests*.

4.7.3 Adaptações

Definir funcionalidades prioritárias para a execução de *smoke tests*. Estratégia de priorização com base nos componentes que sofreram as mudanças mais críticas desde a última entrega.

4.7.4 Contexto Resultante

Mais tranqüilidade e confiança antes de repassar versões intermediárias do software para diferentes times de teste geograficamente distribuídos. Menos desperdício de esforço e redução na demanda por na comunicação entre os times.

4.8 Padrão *Regression Test*

4.8.1 Contexto de Aplicação

Defeitos críticos são encontrados e corrigidos no software. Dependendo de quão críticos, há uma necessidade de garantir que os mesmos não voltarão a se manifestar em uma nova versão.

4.8.2 Problemas enfrentados

Definição de prioridades dos defeitos encontrados no software. Muita comunicação entre times de desenvolvimento no projeto com o propósito de rastrear tais defeitos e propor soluções para os mesmos.

4.8.3 Adaptações

Criação do papel do integrador para cada um dos parceiros. O integrador é responsável por fazer o rastreamento dos *bugs* sob sua responsabilidade e notificar, para cada entrega, quais as correções e pendências. Além disso, o integrador e arquiteto devem fornecer soluções e prazos para os defeitos que se manifestam em uma nova versão do software.

4.8.4 Contexto Resultante

Áreas funcionais de alta prioridade estão protegidas pelos testes. Entretanto, há uma demanda por comunicação para definir qual o nível de regressão desejado em cada execução de testes.

4.9 Padrão *Release Line*

4.9.1 Contexto de Aplicação

Necessidade de se ter uma base única para a geração de versões oficiais do produto. Linhas de codificação dos parceiros são isoladas da linha de codificação de produção, responsável pela geração de *releases*.

4.9.2 Problemas enfrentados

Esforço para a junção das linhas de codificação dos diversos parceiros (processo de integração) e necessidade de verificar a consistência de cada entrega através de um *checklist* desenvolvido pelo gerente de configuração de software. Este *checklist* tem como finalidade a aceitação ou rejeição da entrega do parceiro.

4.9.3 Adaptações

A própria linha de codificação de produção é utilizada como linha para geração de um novo release do produto, evitando o uso de uma linha de codificação.

4.9.4 Contexto Resultante

Capacidade de produzir *releases* do software independentemente de qualquer tarefa de desenvolvimento em andamento pelos parceiros.

4.10 Padrão *Codeline Policy*

4.10.1 Contexto de Aplicação

Diferentes organizações, diferentes culturas, todos envolvidos na construção de um único software. Necessidade de um nível mínimo de uniformidade nas ações a fim de garantir a produtividade coletiva dos desenvolvedores.

4.10.2 Problemas enfrentados

Dificuldade para os desenvolvedores assimilarem todas as idéias contidas na política de gerência da linha de codificação. Divergências entre formas de trabalho, cultura e metodologia de desenvolvimento de software.

4.10.3 Adaptações

Os parceiros receberam um treinamento de gerência de configuração de software no início do projeto. Desta forma, foram definidos e enfatizados os papéis e responsabilidades no processo de SCM.

Cada parceiro é responsável pelo ciclo de vida de sua respectiva linha de codificação (atualizações, *commits*, rotulação e entregas oficiais). Concentração das responsabilidades mencionadas no papel do *integrador*. Cada parceiro possui um desenvolvedor desempenhando o papel especial de integrador.

4.10.4 Contexto Resultante

Redução da sobrecarga e dos ruídos na comunicação no grupo. Agilidade na tomada de decisão a cada ciclo de integração. Facilidade para absorver novos desenvolvedores no projeto. Maior potencial para que todas as linhas de codificação se mantenham mais estáveis ao longo do projeto.

5 Conclusão

O fato desse estudo ter sido realizado num projeto que envolvia equipes de diferentes localidades e de diferentes empresas influenciou fortemente a estruturação das soluções de gerência de configuração para o projeto observado. A necessidade de compartilhar artefatos de software em um contexto global de países e organizações colocou grandes desafios para o projeto. O fator terceirização adiciona mais complexidade ao uso dos

padrões de SCM. O principal motivo são as diferenças culturais e os objetivos dos diferentes parceiros envolvidos.

As soluções foram sendo implementadas de acordo com as necessidades do projeto e o conhecimento da linguagem de padrões proposta por [3]. Deste modo, foi possível identificar os padrões e traçar as devidas correlações. Neste âmbito, a linguagem contribuiu para a reflexão das soluções então estabelecidas. Considerando fatores como organização, arquitetura e *time-to-market*, pode-se também utilizar outras linguagens de padrão, como por exemplo, os padrões organizacionais definidos por [2]. Desta forma, é possível compreender de forma ampla o ciclo de influência entre organização, arquitetura e gerência de configuração.

6 Apêndice "A" – Ferramentas utilizadas

Este apêndice contém informações sobre as ferramentas utilizadas pelo projeto para dar sustentação ao emprego dos padrões de gerência de configuração. As informações estão contidas na tabela 1.

Ferramenta	**Aplicação**
Subversion	Repositório de versões. Criação de branches (linhas de codificação) para permitir o trabalho simultâneo dos diversos parceiros e posterior integração [8].
Kdiff3	Checar se ocorre sobreposição entre o código entregue pelos parceiros. Permite comparar diretamente três fontes de dados [9].
Ant/ make	Automação de tarefas envolvendo o produto do software e os espaços de trabalho (workspace) [10]/[11]: • Geração de builds • Empacotamento de componentes • Instrumentação de código para *profiling* (análise dinâmica) • Invocação das ferramentas para a análise estática do código
Check	Infra-estrutura C para execução de testes automatizados escritos na mesma linguagem [12].
Checkstyle	Análise de padrões de codificação. Detecção de práticas perigoras (*anti-patterns*) [13].
CCCC	Métricas de código: tamanho de módulos e funções em NCSS (*non-commented source statement*), complexidade ciclomática por função. Suporte à C/C++ [14].
JavaNCSS	Análogo ao CCCC, só que para Java [15].
Simian	Analisador estático de redundância no código. Suporta múltiplas linguagens: C/C++, Java, C#, entre outras [16].

RPM/ JAR	Padrões para empacotamento de componentes e produtos de software [17]/[18].
Unix tools	grep, sed, find, bash scripts, etc.

Tabela 1 – Ferramentas empregadas na aplicação dos padrões de SCM

7 Referências

[1] Osterweil, L., *Software Processes are Software Too,* ACM, 1987.

[2] Coplien, J., Harrison N., *Organizational Patterns for Agile Software Development*, Prentice Hall, 2004.

[3] Berczuk, S., Appleton, B., *Software Configuration Management Patterns,* Addison-Wesley, 2002.

[4] Bass, L., Kazman, R., Clements, P., *Software Architecture In Practice*, SEI Series in Software Engineering, 2002

[5] McConnel, S., Rapid Development, Microsoft Press, 1996

[6] Fowler, M., *Continuous Integration*, http://www.martinfowler.com/articles/continuousIntegration.html, última visita em [09/07/2005].

[7] Gabriel, R., *Software Patterns*, Oxford Press, 1996

[8] Subversion, http://subversion.tigris.org/, última visita em [10/07/2005].

[9] Kdiff3, http://kdiff3.sourceforge.net/, última visita em [10/07/2005].

[10] Apache Ant, http://ant.apache.org/, última visita em [10/07/2005].

[11] GNU make, http://directory.fsf.org/make.html, última visita em [10/07/2005].

[12] Check, http://check.sourceforge.net/, última visita em [10/07/2005].

[13] Checkstyle, http://checkstyle.sourceforge.net/, última visita em [10/07/2005].

[14] CCCC, http://cccc.sourceforge.net/, última visita em [10/06/2005].

[15] JavaNCSS, http://www.kclee.de/clemens/java/javancss/, última visita em [10/06/2005].

[16] Simian, http://www.redhillconsulting.com.au/products/simian/, última visita em [10/06/2005].

[17] RPM, http://www.rpm.org/, última visita em [10/06/2005].

[18] JAR, http://java.sun.com/j2se/1.5.0/docs/guide/jar/, última visita em [10/06/2005].