Utilização de Sistemas de Controle de Versão para facilitar o desenvolvimento colaborativo de software

André Silveira Wesley Dal Col

5 de maio de 2010

RESUMO

De modo a atender às diversas demandas do mercado, é cada vez mais comum que o desenvolvimento de software seja realizado 24 horas por dia, por equipes distintas trabalhando de forma descentralizada ao redor do globo. Por isso, são cada vez mais necessárias ferramentas que propiciem o rastreamento e a integração do código fonte. Alguns sistemas de cotrole de versão que foram desenvolvidos para auxiliar na primeira tarefa, evoluíram, tornando-se também boas ferramentas de programação integrado-colaborativa. O foco deste trabalho é extensão do uso dessas ferramentas para auxiliar o desenvolvimento colaborativo entre equipes geograficamente distantes.

Palavras-chave: Controle de Versão, Colaboração, Integração, Desenvolvimento de Software

ABSTRACT

In order to meet various market demands, it is increasingly common for software development is carried out 24 hours a day, by separate teams working in a decentralized way around the globe. Therefore, they are increasingly necessary tools that provide tracking and integration of source code. Some version control systems that were developed to assist in the first task, evolved, becoming too well-integrated programming tools collaboratively. The focus of this paper is to extend the use of these tools to support collaborative development among geographically dispersed teams.

Palavras-chave: Version Control, Colaboration, Integration, Software Development

SUMÁRIO

1 INTRODUÇÃO

Nos primórdios da computação os computadores eram manipulados apenas por um pequeno grupo de técnicos altamente especializados, que precisavam plugar uma grande quantidade de cabos nos enormes painéis que compunham os computadores da época, num processo lento e muito suscetível a erros.

Com o desenvolvimento da tecnologia e o advento das linguagens e técnicas de programação modernas, a tarefa de criar programas de computador pode ser desempenhada por um número maior de pessoas, muitas vezes, cada uma sendo responsável por apenas uma fração de todo o sistema. Unir todas essas partes e garantir que elas funcionem juntas não é uma tarefa simples. Analogamente a correção manual de erros em grandes sistemas de computação é frequentemente uma tarefa tão dispendiosa quanto criá-lo e mantê-lo funcionando.

Para resolver esses problemas de forma automatizada, com o passar dos anos algumas ferramentas foram sendo desenvolvidas, como os programas diff e patch que facilitam processo de aplicar uma correção em uma determinada parte do software, assim como os Sistemas de Controle de Versão dos quais podemos citar o CVS, o Subversion e Git, que são utilizados para rastrear e controlar a evolução do código fonte, o que torna o trabalho das equipes de desenvolvimento um pouco menos complicado.

Entretanto, a dinâmica atual do mercado de tecnologia gera demandas cada vez mais agressivas, fazendo com que o desenvolvimento de software seja feito quase que 24 horas por dia, muitas vezes por equipes geograficamente distantes ou em fuso-horários muito diferentes, ressurgindo o desafio de integrar vários componentes de software agora produzidos não somente por pessoas distintas mas por empresas e instituições das mais diversas naturezas e culturas, com o máximo de qualidade possível minimizando o tempo de desenvolvimento e os custos.

Com projetos compostos por milhões de linhas de código, torna-se impraticável que a integração entre as partes seja realizada manualmente, pois basta somente um equívoco para provocar erros muito difíceis de corrigir.

Para tentar resolver esse e outros problemas novos Sistemas de Controle de Versão foram criados, melhorando os mecanismos de mesclagem de código e buscando uma arquitetura descentralizada, de forma que as equipes possam trabalhar com um mínimo de interferência entre si, enquanto a etapa de integração é feita cada vez mais de forma automática.

1.1 OBJETIVOS

O objetivo principal deste trabalho é demonstrar como os Sistemas de Controle de Versão modernos podem ser utilizados para facilitar o desenvolvimento de software colaborativo, explorando duas de suas principais características, a arquitetura descentralizada e a facilidade com que integram peças de código.

Será feito um breve levantamento histórico contextualizando alguns dos principais Sistemas de Controle de Versão em uso atualmente.

Em seguida demostrar-se-á como a arquitetura descentralizada facilita o desenvolvimento de ramificações independentes do código fonte com um mínimo de conflitos entre elas, confrontando esse cenário com situações onde não há ferramenta de controle de versão ou quando a ferramenta utilizada não permite essa abordagem.

Também será mostrado como as novas técnicas de rastreamento da genealogia de um projeto de software permitem uma integração muito mais suave e automática do que quando são utilizadas ferramentas menos sofisticadas ou nenhuma ferramenta.

2 HISTÓRICO

Neste capítulo será abordado brevemente um pouco da história do versionamento de arquivos no contexto do desenvolvimento de software, quais foram as necessidades de cada época e quais as ferramentas desenvolvidas para supri-las, mostrando como foi a evolução deste processo ao longo dos anos.

2.1 A AURORA DAS LINGUAGENS DE PROGRAMAÇÃO MODERNAS

No século XIX teares programáveis e tocadores automáticos de piano já implementavam o que hoje é conhecido por Linguagem de Domínio Específico (DSN - Domain Specifc Language).

No início do século XX o formalismo dos trabalhos de Alonzo Church (Cálculo Lambda) e de Alan Turing (Máquina de Turing) foreneceram as bases matemáticas necessárias para se expressar Algoritmos.

Entretanto, as primeiras linguagens de programação utilizadas por computadores digitais só foram criadas a partir da década de 1940. O termo portabilidade apareceu na década de 1950 e a partir da década de 1960 começaram a ser desenvolvidos os principais paradigmas de programação que conhecemos hoje.

De lá para cá a popularização do uso de computadores fez com que a demanda por software crescesse absurdamente, criando toda uma indústira que movimenta bilões de dólares anualmente.

Acompanhado essas cifras, bilhões de linhas de código tem de ser produzidas, mantidas e atualizadas constantemente, o que nunca foi uma tarefa simples e fica cada vez mais complexa a medida que os sistemas crescem.

2.2 AS FERRAMENTAS DIFF E PATCH

Até a década de 1970, qualquer correção feita em um programa era feita em uma cópia do arquivofonte original (se disponível) e encaminhada ao auto do software para que ele analisasse as alterações e incorporasse no conjunto oficial de fontes.

A tarefa de procurar alterações dentro dos enormes arquivos de código fonte era então realizada manualmente pelo programador, linha a linha, num processo bastante lento.

Em 1974 a primeira versão do programa diff foi liberada juntamente com a quinta versão do sistema operacional Unix, produzido no Bell Laboratories, por Douglas McIlroy, basead em um protótipo escrito por James W. Hunt da Universidade de Stanford. Em 1976 eles publicaram um artigo sobre o algoritmo utilizado.

O programa diff aceita como entrada arquivos ou diretórios e gera como saída apenas as diferenças entre eles especificando os arquivos e números de linhas onde elas ocorrem. O formato da saída ficou conhecido pelo nome diff, e posteriormente patch.

O trabalho de encontrar as diferenças entre os arquivos foi simplificado, mas a tarefa de aplicar as correções ainda era realizada manualmente.

Para resolver esse problema, em maio de 1985 Larry Wall criou o programa patch. Esse programa pega um arquivo no formato gerado pelo diff e aplica no arquivo completo de fontes, automatizando a tarefa de correção. Logo esse processo passou a ser conhecido como patching, onde criar um patch é gerar um arquivo no formato diff e aplicar o petch é usar o programa patch para implantar as alterações.

Esses dois prgramas facilitaram em muito o trabalho de programadores que precisavam gerenciar várias atualizações e correções, entretanto, uma vez aplicado o patch, removê-lo não é tarfa simples. Isso podia acontecer quando por exemplo, um arquivo de diff ou de fonte errados eram utilizados, ou quando a aplicação de um patch corrigia uma falha, mas causava outras.

O aumento do número de patches trouxe outra demanda, a necessidade de saber quem aplicou qual patch e quando, bem como desfazer as alterações caso necessário.

2.3 O NASCIMENTO DO CONTROLE DE VERSÕES

Uma das mais utilizadas ferramentas de controle de revisão dessa época foi o RCS, criado em 1982 por Walter F. Tichy, sendo uma das primeiras a automatizar tarefas de armazenar,

recuperar, identificar e mesclar revisões.

Embora razoavelemente útil para lidar com alguns arquivos de texto o RCS ainda não supria várias necessidades no desenvolvimento de software.

Como um conjunto de scripts trabalhando em conjunto com o RCS, surge em julho de 1986 o CVS, criado por Dick Grune. Em 1989, Brian Berliner o reescreveu completamente em C e Jeff Polk adicionou algumas funcionalidades posteriormente (**??**).

Mesmo sendo uma ferramenta bastante sofisticada para a época, o CVS ainda mantinha os principais problemas do RCS, como só poder ser utilizado localmente, o que só foi resolvido na década de 1990 por Jim Kingdon.

> I created CVS to be able to cooperate with my students, Erik Baalbergen and Maarten Waage, on the ACK (Amsterdam Compiler Kit) C compiler. The three of us had vastly different schedules (one student was a steady 9-5 worker, the other was irregular, and I could work on the project only in the evenings). Their project ran from July 1984 to August 1985. CVS was initially called cmt, for the obvious reason that it allowed us to commit versions independently.
>
> – Dick Grune, Dick Grune's website

O CVS foi construído para auxiliar equipes nas quais os programadores tem de trabalhar em horários diferentes, mas um por vez. Quando surge a necessidade de vários programadores trabalharem simultaneamente no mesmo projeto, o design do CVS já não atende tão bem.

Em 1995 Karl Fogel e Jim Blandy criaram a emrpesa Cyclic Software oferecendo suporte ao CVS, e embora a tenham vendido a empresa posteriormente, continuavam a utilizar o CVS no seu dia a dia. Suas frustrações com os problemas do CVS foram tais que Jim começou a pensar em uma melhor forma de lidar com versioanamento.

No início de 2000, a CollabNet contratou Karl para desenvolver um novo sistema de controle de versão para substituir o CVS na sua suite colaborativa CollabNet Enterprise Edition (CEE). Jim então convenceu a empresa onde trabalhava, a Red Hat, a cedê-lo para esse projeto, iniciando assim o desenvolvimento do Subversion (**??**).

O Subversion foi então um dos primeiros sistemas de controle de versão open source a pensar desde o seu projeto no desenvolvimento colaborativo.

Embora o Subversion tenha sido desenhado para suprir diversas demandas não satisfeitas pelo CVS, ele ainda era fortemente influenciado pelo seu antecessor, mantendo por exemplo

uma arquitetura centralizada, onde a figura de um servidor central era necessária durante a operação.

2.4 O LINUX E O CONTROLE DESCENTRALIZADO DE VERSÕES

O Linux é de longe um dos projetos de software livre de maior sucesso na atualidade, e talvez um dos maiores difusores da filosofia open source.

Liberada em 1991 em um post no newsgroup "comp.os.minix." como o resultado de um hobby, a primeira versão do Linux logo cresceu, ganhando novos contribuidores e tornando-se um dos principais sistemas operacionais utilizados em servidores.

> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).
>
> I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)
>
> Linus (torvalds@kruuna.helsinki.fi)
>
> PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.
>
> Linus Torvalds, Usenet group comp.os.minix, August 25, 1991.

Contando com a ajuda de programadores de todas as partes do mundo, logo foi necessário utilizar um sistema de controle de versão capaz de gerenciar toda a complexidade por trás de um desenvolvimento descentralizado.

Para esta tarefa Linus utilizou por vários anos o BitKeeper, ferramenta proprietária mas que ao contrário do CVS e SVN possui uma arquitetura distribuída, facilitando o trabalho de colaboradores espalhados mundo afora.

Mas em 2005 o licenciamento do BitKeeper mudou, tornando o seu uso no projeto Linux inviável. Linus então decidiu criar um sistema de controle de versão descentralizado tomando como base alguns critérios:

1. Tomar o CVS como um exemplo do que não fazer; na dúvida, fazer exatamente a escolha oposta ao que o CVS fez.
2. Suportar um workflow parecido com o do BitKeeper.
3. Fortíssima segurança contra corrupção, acidenta ou maliciosa
4. Altíssima performance

3 DESENVOLVIMENTO COLABORATIVO

3.1 APLICANDO PATCHES

3.2 MODELOS DE CICLO DE TRABALHO

3.2.1 LOCK-MODIFY-UNLOCK

3.2.2 COPY-MODIFY-MERGE

3.2.3 INTEGRATION MANAGER

3.2.4 DICTATOR AND LIEUTENANTS