Gerência de Configuração de Software

1)

Haviam problemas em relação ao desenvolvimento e manutenção de software por conta da falta de organização das modificações efetuadas concomitantemente por diferentes desenvolvedores, tais como o porquê, determinado local e o que foi feito para corrigi-la.

A Gerência de Configuração de Software é uma disciplina que controla e notifica as inúmeras correções, extensões e adaptações aplicadas durante o ciclo de vida do software de forma a assegurar um processo de desenvolvimento e evolução sistemático e rastreável, sendo indispensável quando equipes manipulam, muitas vezes em conjunto, artefatos comuns.

É uma disciplina que controla e notifica as inúmeras correções, extensões e adaptações aplicadas durante o ciclo de vida do software de forma a assegurar um processo de desenvolvimento e evolução sistemático e rastreável, sendo indispensável quando equipes manipulam, muitas vezes em conjunto, artefatos comuns.

2)

Versionamento de código surgiu para corrigir e organizar as versões de códigos em manutenção ou desenvolvimento, permitindo que enquanto o código está sendo desenvolvido, o programador vai criando as versões conforme melhora, muda ou implementa algo. Depois, **é possível voltar às versões anteriores** caso seja necessário.

Por conta disso, o versionamento de código faz parte das boas práticas de programação e se torna **importante para a organização do projeto**, consegue contribuir com o **trabalho em equipe** e também criar um “backup” de todas as mudanças.

3)

Existem algumas ferramentas de controle de versões, como o **Git**que é a mais conhecida, contando com várias funcionalidades que auxiliam nessa organização. Mas também existem outras, como SVN e CVS.

**Sistemas de Versionamento Distribuídos:**

o Git, assim como Mercurial e Bazaar, não existe um repositório central. É claro que você pode eleger um como tal, mas cada repositório, mesmo o da máquina do desenvolvedor, contém uma cópia completa e funcional do repositório.

Uma desvantagem desse modelo é que a clonagem inicial do repositório pode demorar bastante, já que não será feita a transferência apenas da cópia atual de cada artefato, mas também do histórico, *tags* e *branches*. Algo que pode minimizar isso é a possibilidade de recuperar partes do repositório seletivamente, como *branches*, tags ou mesmo por data. Mas não sei de detalhes sobre até que ponto isso é implementado em cada um dos sistemas.

Outra desvantagem é a dificuldade de um gerenciamento centralizado e um controle de acesso efetivo, já que os repositórios ficam distribuídos em vários ambientes.

Além disso, nos sistemas de versionamento distribuídos, commit e checkout são feitos no repositório local de cada ambiente. Após a conclusão do trabalho, com a classe devidamente "commitada", tags "passadas" e "branches mergeadas", o desenvolvedor precisa sincronizar seu repositório local com o repositório remoto. Isso é feito com os comandos push (envia atualizações do seu repositório local para o remoto) e pull (recupera atualizações do repositório remoto para o local).

É um pouco mais complicado trabalhar com sistemas de versionamento distribuídos, mas as vantagens são muitas:

* Exceto o pull inicial, eles são muito mais rápidos do que os sistemas centralizados como CVS e SVN.
* Muitas operações não necessitam de acesso à rede, então o desenvolvedor pode trabalhar *offline*, sincronizando com o repositório remoto apenas quando necessário.
* O desenvolvedor pode trabalhar em modo privado, gerando tags, branches e versões que serão simplesmente descartadas.
* Exceto quando há conflitos, o merge é automático.
* Cada cópia do repositório funciona como um backup do repositório "principal".

Note que existem diferenças entre os sistemas de versionamento distribuídos. Não conheço quase nada do Bazaar, mas posso citar alguns casos interessantes com relação ao Git e o Mercurial:

* O comando git pull inclui o update e atualiza os arquivos em uso do projeto, então o pull do Git é diferente do conceito tradicional de pull, sendo equivalente a pull + update. O pull puro seria na verdade git fetch.
* No Git, você precisa adicionar manualmente os arquivos novos e alterados ao *staging area* com o comando git add para serem "commitados". O Mercurial faz isso automaticamente por padrão.

**GIT:**

Com ela, é possível criar versões detalhadas, além de outras funções presentes que são úteis no dia a dia.

É uma ferramenta tão importante que muitas empresas incluem o conhecimento em Git e versionamento de código como **pré-requisito para vagas em desenvolvimento**.

O **Git funciona localmente, ou seja, na máquina do programador**e organiza o código e suas versões enquanto o desenvolvimento está sendo feito. Assim, oferece diferentes ferramentas e ações que o programador utiliza para tornar suas versões mais organizadas.

Depois, é possível subir toda essa organização para um repositório remoto, o que é bastante comum no dia a dia empresarial. Isso porque, na prática, são vários programadores mexendo no mesmo código, cada um em sua máquina. Depois, reúne-se tudo remotamente.

É um controle de versão bastante diferente do CVS e SVN, pois o modelo de versionamento é descentralizado (não há exatamente um fluxo central, e quando ele existe, não deve ser alterado, recebendo apenas *merges* de outros fluxos de desenvolvimento) e o envio de arquivos é em duas fases:

1. **Commit**, fase em que as alterações são armazenadas apenas localmente;
2. **Push**, fase em que as alterações são enviadas para um servidor que concentra todos os conjuntos de alterações (chamados de *changesets*) que podem ser recombinados entre si livremente.

Como o GIT não é um servidor de arquitetura linear, é ideal para projetos em início de desenvolvimento em que conflitos são comuns e funcionalidades são desenvolvidas em separado. O processo de *merge* é o mais completo e tolerante dentre os três.

**Sistemas de Versionamento Centralizados**

CVS e SVN possuem um repositório central de onde os usuários fazem o checkout e commit dos artefatos versionados.

A vantagem dessa abordagem é que você pode ter um controle central sobre os projetos, impor segurança de acesso mais facilmente. Além disso, há a possibilidade de bloquear arquivos (*lock*).

Porém, existem muitas desvantagens. A principal delas é que esse tipo de sistema não escala muito bem, isto é, muitas equipes e projetos no mesmo repositório tendem a deixá-lo lento. Outra desvantagem importante é que os usuários não podem fazer muita coisa *offline*, sendo necessário sempre estar conectado ao servidor central para realizar operações como criar *tags*, *branches*, fazer *merge*, etc.

Além disso, existem diferenças significativas entre CVS e SVN:

* SVN consegue rastrear arquivos renomeados.
* Se versionamento centralizado é lento, o CVS consegue ser mais ainda.
* O commit do CVS é por arquivo. Já o SVN consegue agrupar as mudanças de um commit, então é possível por exemplo voltar a uma revisão anterior. Isso facilita muito encontrar qual commit quebrou o código.

**CVS:**

Foi um dos primeiros sistemas de controle de versão a ter fluxos alternativos de desenvolvimento e a permitir livre edição de arquivos somente-texto entre várias pessoas de um mesmo time ao mesmo tempo.

A ideia de um repositório dentro das normas do CVS é no esquema de árvore, compreendendo um fluxo principal de desenvolvimento (chamado de trunk, ou *tronco* em português), fluxos alternativos de desenvolvimento (chamados de branches, ou *galhos*), em que são implementadas mudanças em separado do fluxo principal, e tags (*etiquetas*, que são revisões dos outros dois fluxos que não podem mais ser alteradas, ideais para indicar versões estáveis).

**SVN:**

É um sistema de controle de versão open-source que gerencia arquivos e diretórios controlando as alterações realizadas ao longo do tempo. Além disso, é possível recuperar versões anteriores ou visualizar o histórico de alterações.

É a evolução do modelo do CVS que resolve bem várias limitações do CVS, como a introdução dos comandos Rename e Move, que não apenas renomeia/move o arquivo como mantém seu histórico de alterações, o comando Commit (de envio de arquivos) ser verdadeiramente atômico, suportando *rollbacks* em caso de falhas e versionamento de arquivos não suportados pelo CVS, como links simbólicos.

Possui todos os comandos do CVS e mais alguns:

* Rename
* Move

Possui também capacidade de guardar metadados dos arquivos e diretórios (extensões ignoradas, histórico de merges, etc.).