**Gerencia de Configuração**

A Gerência de Configuração de Software por sua vez é uma disciplina que identifica a configuração de um sistema em diferentes pontos no tempo com a finalidade de **controlar sistematicamente** as mudanças realizadas, mantendo a integridade e rastreabilidade da configuração através do ciclo de vida do sistema.

Configuração de software se trata de uma: característica funcional e física do hardware ou software, conforme estabelecido na documentação técnica ou obtido em um produto.

Uma desvantagem dos sistemas de controle de versão que adotam o modelo lock-modify-unlock é a baixa produtividade em trabalhos colaborativos: esse modelo não permite que o conteúdo de um arquivo seja alterado por mais de um usuário simultaneamente.

Muitos sistemas de controle de versão usam o modelo lock-modify-unlock (travar-modificar-destravar) para resolver o problema de vários autores destruírem o trabalho uns dos outros. Neste modelo, o repositório permite que apenas uma pessoa de cada vez altere o arquivo. Essa política de exclusividade é gerenciada usando locks (travas). Harry precisa “travar” (lock) um arquivo antes que possa fazer alterações nele. Se Harry tiver travado o arquivo, então Sally não poderá travá-lo também, e, portanto, não poderá fazer nenhuma alteração nele. Tudo que ela pode fazer é ler o arquivo, e esperar que Harry termine suas alterações e destrave (unlock) o arquivo.

**GIT**

O Git, sistema de controle de versões que mantém um histórico completo de todas as alterações, permite a recuperação das versões do projeto na busca de informações sobre o estado dos arquivos em versões anteriores. Este é o conceito de todo sistema de Controle de Versão: cuida das mudanças feitas nos artefatos, salvando-o em um repositório que pode ser acessado (check-out), modificado e depois, atualizado (check-in). Cada atualização é uma nova versão do artefato, **mantendo histórico das versões e suas mudanças.**

O workflow básico do Git pode ser descrito assim:  
  
1. Você modifica arquivos no seu diretório de trabalho.  
2. Você seleciona os arquivos, adicionando snapshots deles para sua área de preparação.  
3. Você faz um commit, que leva os arquivos como eles estão na sua  
área de preparação e os armazena permanentemente no seu diretório Git.

Git é um sistema de versionamento, ele considera que os dados são como um  
conjunto de ***snapshots*** (captura de algo em um determinado instante, como em  
uma foto) de um minissistema de arquivos. Cada vez que você salva ou consolida  
(commit) o estado do seu projeto no Git, é como se ele tirasse uma foto de  
todos os seus arquivos naquele momento e armazenasse uma referência para essa  
captura. Para ser eficiente, se nenhum arquivo foi alterado, a informação não é  
armazenada novamente - apenas um link para o arquivo idêntico anterior que já  
foi armazenado. A figura 1-5 mostra melhor como o Git lida com seus dados.

SVN

Não se faz alterações na tag, mas sim no branch*.*

***TAG*** *-* É um marcador de um estado do código em um determinado momento. É um ponto no tempo no *trunk* ou em um *branch* que você deseja preservar. As duas principais razões para a preservação seriam:

* Este é um grande lançamento do software, se alfa, beta, RC ou RTM;
* Este é o ponto mais estável do software antes de aplicar revisões importantes sobre o *trunk*.

Não é comum trabalhar em cima de uma *tag*. É criado um marco que pode ser acessado facilmente. Quando encontrar um *bug* em versão antiga que precisa de uma solução, é fácil criar um *branch* em cima dele para fazer o conserto. O que costuma diferenciar *tag* de *branch* é justamente a estabilidade do conteúdo. Você não deve mexer em um repositório *tag*. Ele se diferencia do *trunk* por ser algo secundário e quase sempre está no passado.

**Conceitos diversos**

* **Integração Contínua (Continuous integration)** - As metodologias ágeis de desenvolvimento permitiram entregas rápidas com pequenas e frequentes modificações nos códigos, em vez de versões criadas em períodos longos de tempo com grandes alterações. Com essa otimização, logo o trabalho em paralelo dos membros da equipe de desenvolvimento tornou-se possível. Como as entregas são rápidas, um programador pode atuar em uma parte do código enquanto os demais em outras e assim o desenvolvimento é mais ágil. Integração Contínua é um processo essencial dessas metodologias ágeis que permite a integração do trabalho dos membros de uma equipe o mais rápido possível com a execução de builds e testes automatizados do código.
* **Entrega Contínua (Continuous Delivery)** - A Entrega Contínua é um conjunto de práticas com o objetivo de garantir que um novo código esteja apto para ser disponibilizado em ambiente de produção. No entanto, o deploy em ambiente de produção não é automático, sendo é uma decisão de negócio a ser aprovada previamente.
* **Implantação Contínua (Continuous Deploy)** - Esta prática é o próximo passo da Entrega contínua. Assim que o programador julga pronto seu código e aciona a solicitação para deploy, são realizadas todas validações previstas nas metodologias anteriores e, se não houverem falhas, o novo código é disponibilizado automaticamente em ambiente de produção.