UFES

Engenharia de Software

Unidade 10: Gerência de Configuração de Software

Prof. Fabrício Martins Mendonça

Alegre, ES

Objetivos

- Apresentar os conceitos básicos do gerenciamento de configuração no desenvolvimento de software.
- Descrever os mecanismos utilizados no gerenciamento de mudanças e de controle de versões de software.
- Entender a importância desses mecanismos em contextos ágeis, sujeitos a muitas mudanças.
- Apresentar ferramentas para o gerenciamento de mudanças e controle de versões do software.

Como visto antes...

- Independente do domínio de aplicação, tamanho ou complexidade o software continuará a **evoluir** com o tempo, pressionado e dirigido **pelas mudanças**.

- Leis de Lehman (1997) apud Pressman (2011)
    - Exemplo: Lei da Mudança Contínua

- Atualmente a **volatilidade** dos requisitos de software é mais uma regra do que uma exceção e as metodologias ágeis lidam com essa **imprevisibilidade.**

Como visto antes...

- Algumas razões das mudanças no software são:
  - ✓ Erros e defeitos identificados;
  - ✓ Alterações das funcionalidades atuais;
  - ✓ Novas funcionalidades;
  - ✓ Novas tecnologias;
  - ✓ Novos negócios ou condições de mercado;
  - ✓ Crescimento ou enxugamento do sistema;
  - ✓ Restrições orçamentárias ou de cronograma.

Introdução

- Porque os softwares mudam frequentemente, os sistemas podem ser pensados como um conjunto de versões, e cada qual precisa ser mantida e gerenciada.

- Versões implementam propostas de mudanças, correções de defeitos, e adaptações de hardware e sistemas operacionais diferentes. É fácil perder a noção de quais mudanças e versões foram incorporadas no sistema.

- O gerenciamento de configuração (CM – Configuration Management) se interessa pelas políticas, processos e ferramentas para o gerenciamento de sistemas de software que sofrem mudanças.

Gerência de Configuração de Software

- *Software Configuration Management* (SCM) é a parte da engenharia de software responsável por identificar, organizar e controlar modificações no software com o objetivo de maximizar a produtividade e minimizar os erros (Babich, 86).

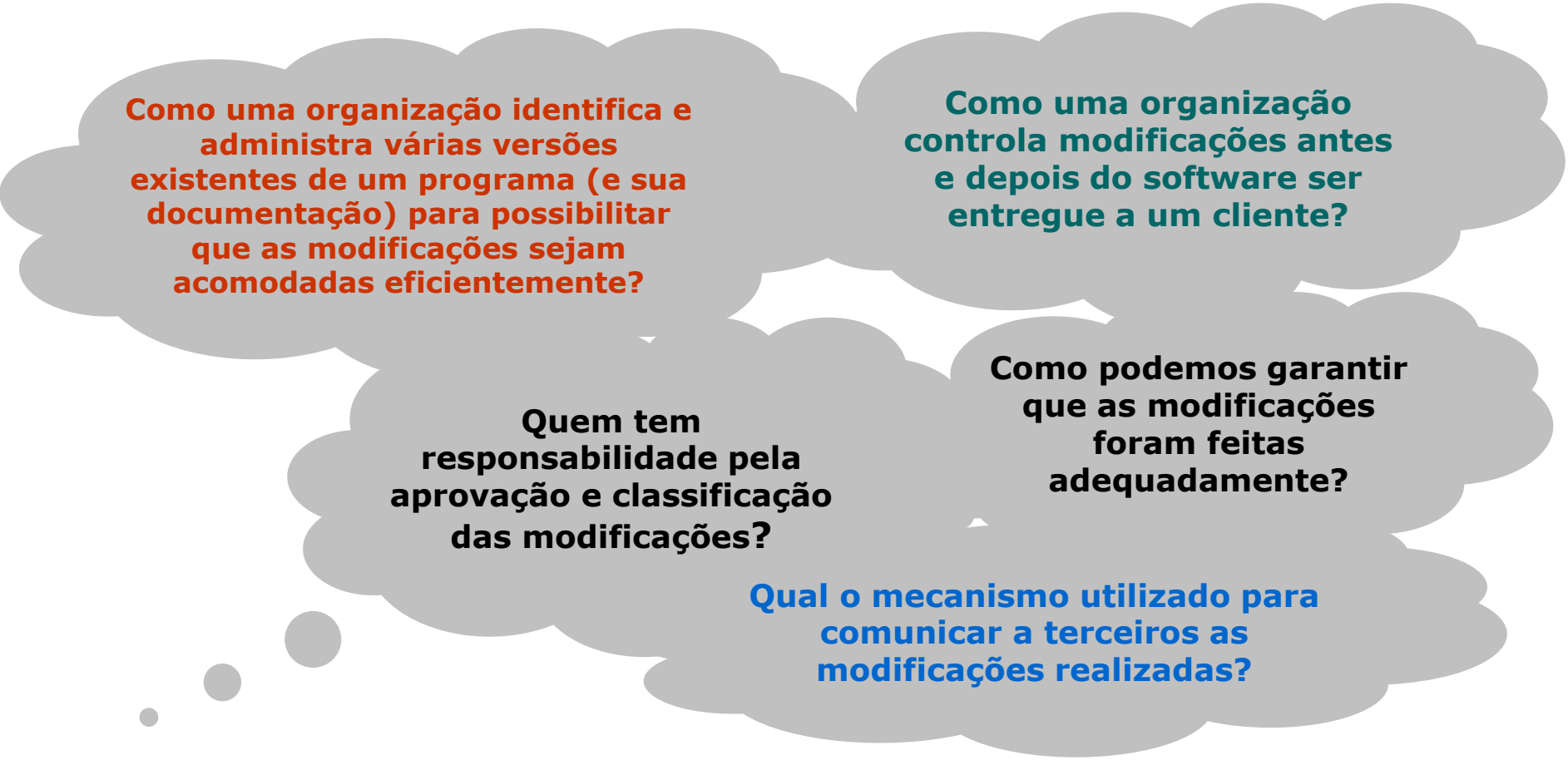
- SCM é parte essencial da gestão de qualidade.

- O CMMI-SW insere a gerência de configuração como uma das 7 áreas-chave para que uma empresa possa alcançar o nível 2 de maturidade: nível gerenciado.

- “Controle as alterações de software, senão elas irão controlar você (o processo)” (Pressman, 2011).

Gerência de Configuração de Software

- Questões Complexas da Gerência de Configuração



Atividades de SCM

- **Gerenciamento de mudanças**
  - ✓ Manter o acompanhamento das solicitações de mudanças no software dos clientes e desenvolvedores, definir os custos e o impacto das mudanças, e decidir quais mudanças devem ser implementadas.

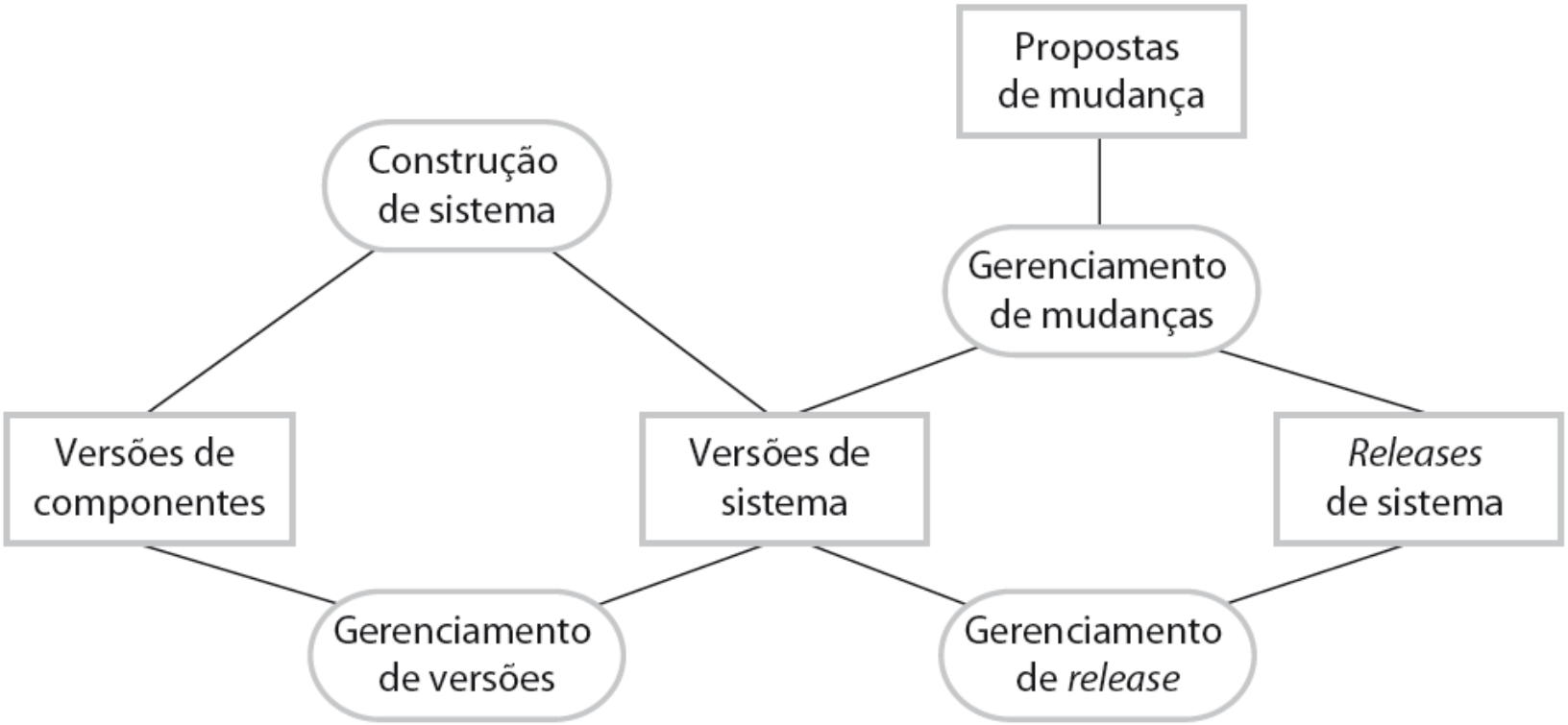
- **Gerenciamento de versões**
  - ✓ Manter o controle das múltiplas versões de componentes do sistema e assegurar que as alterações feitas aos componentes por diferentes desenvolvedores não interfiram umas com as outras.

Atividades de SCM

- **Construção do sistema**
  - ✓ O processo de montagem dos componentes de programa, dados e bibliotecas, e em seguida, a compilação desses para criar um sistema executável.

- **Gerenciamento de releases**
  - ✓ Preparar o software para release externo e manter o acompanhamento das versões do sistema que foram liberadas para uso pelo cliente.

Atividades de SCM



Gerenciamento de Mudanças

- O gerenciamento de mudanças consiste em acompanhar **o processo de mudança**, minimizando seus impactos e mantendo a qualidade dos serviços.

- É uma das 5 áreas-chaves de suporte de serviços do ITIL v2 (2007), mantida na ITIL v3 (2011).

- **Metas** do gerenciamento de mudanças:
  - ✓ Garantir que a evolução do sistema seja um processo gerenciado;
  - ✓ Dar a prioridade às mudanças mais urgentes e com melhor relação custo-benefício;
  - ✓ Acompanhar as alterações nos componentes

Gerência de Configuração de Software

- As tarefas envolvidas na gerência de configuração são:

1. Identificar de forma única os artefatos de software produzidos (objetos de configuração);
2. Criar mecanismos de controle e gestão de mudanças;
3. Criar mecanismos de controle de versão;
4. Montar ou configurar o sistema;
5. Gerenciar releases (versões distribuídas);
6. Realizar auditoria do processo de mudança.

Gerência de Configuração de Software

- Papeis envolvidos no processo de SCM:

Papel	Responsabilidades	Visão de SCM
Gerente de Projetos	Gerencia à equipe de software nas atividades de SCM.	Mecanismo de auditoria.
Gerente de Configuração	Assegura o cumprimento dos procedimentos e políticas de SCM.	Mecanismo de controle, de rastreamento e criador de políticas.
Engenheiros de Software	Desenvolvem usando os procedimentos de SCM definidos, resolvendo conflitos de versões.	Mecanismo de alteração, criação e controle de acesso
Cliente/Stackholders	Utilizam o software com mais segurança.	Mecanismo de garantia de qualidade

Gerência de Configuração de Software

Conceitos Básicos

- **Item ou objeto de configuração de software:**
  - qualquer componente ou artefato do processo de software que necessita ser configurado para se entregar um serviço de TI.

- **Repositório:**
  - um armazenamento compartilhado (espécie de BD) de itens de configuração que deve garantir integridade, segurança, backup e acesso concorrente aos itens.

- **Versão:** cada vez que o item é alterado, dizemos que temos uma nova versão ou revisão do mesmo.

Sistemas de Gerenciamento de Versões

- Objetivos principais do gerenciamento de versões:
  - Manter o controle das múltiplas versões de componentes do sistema
  - Assegurar que as alterações feitas aos componentes por diferentes desenvolvedores não interfiram umas com as outras.
- Deve-se garantir o **controle de acesso**: quem tem acesso para acessar e modificar objetos.
- E o **controle de sincronização**: alterações paralelas não sobrescrevem umas às outras.

Sistemas de Gerenciamento de Versões

- **Identificação de versão e release**
  - ✓ Versões gerenciadas recebem identificadores quando são submetidos ao sistema.

- **Gerenciamento de armazenamento**
  - ✓ Devem oferecer recursos de gerenciamento de armazenamento que reduza o espaço de armazenamento exigido por múltiplas versões de componentes que diferem apenas ligeiramente

- **Registro de histórico de mudanças**
  - ✓ Todas as mudanças feitas no código de um sistema ou componente são registradas e listadas.

Sistemas de Gerenciamento de Versões

- **Desenvolvimento independente**
  - ✓ Deve manter o acompanhamento de componentes que foram retirados para edição e garante que as mudanças feitas em um componente por diferentes desenvolvedores não interfiram umas nas outras.

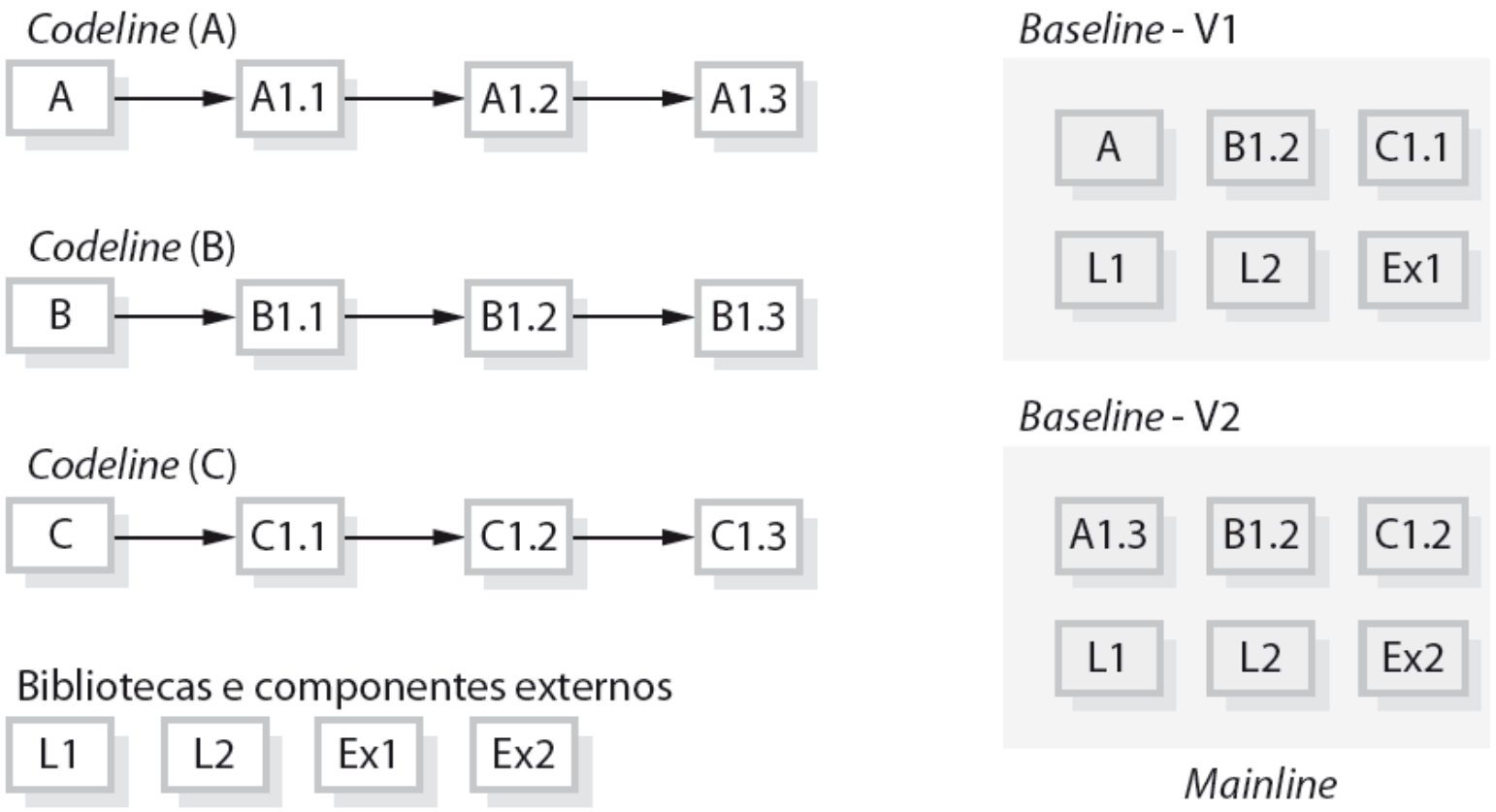
- **Suporte a projetos**
  - ✓ Um sistema de gerenciamento de versões pode apoiar o desenvolvimento de vários projetos que compartilham componentes.

Gerenciamento de Versões

TERMINOLOGIA

- **BaseLine:**
  - Uma versão do sistema estabelecida a partir de algum marco do projeto. Definição de um sistema específico.
- **CodeLine:**
  - Sequencia de versões de código-fonte de um componente de software e outros itens do quais esse componente depende.
- **Workspace:**
  - Área de trabalho privada em que o software pode ser alterado sem afetar outros desenvolvedores.
- **Branching:**
  - criação de uma nova área no repositório a partir de outra já existente.

Gerenciamento de Versões



- *Baselines* são importantes porque muitas vezes você tem de recriar uma versão específica de um sistema completo.

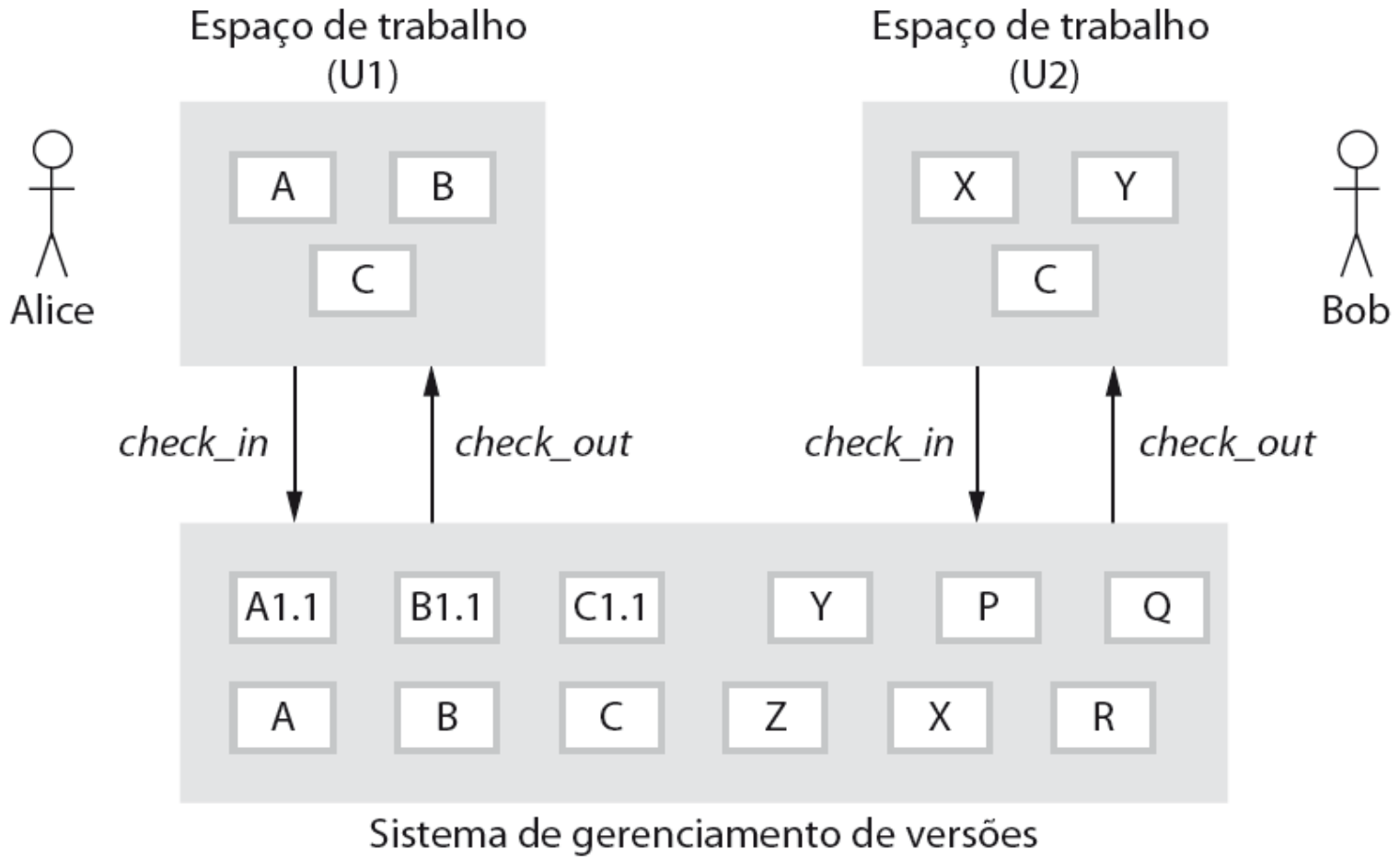
Gerenciamento de Versões

TERMINOLOGIA

- **MainLine ou Truck:**
  - Área principal do repositório onde se encontra o código estabilizado.
- **Check-in ou Commit:**
  - Ação de enviar para o repositório os arquivos que foram alterados em sua cópia de trabalho local.
- **Check-out ou Update:**
  - Ação de atualizar seu workspace local com arquivos alterados no repositório em sua última versão.
- **Merging:**
  - Ação de reintegrar um branching ou atualizá-lo com alterações feitas no branching original.

Gerenciamento de Versões

- *Check-in* e *check-out* a partir de um repositório de versões

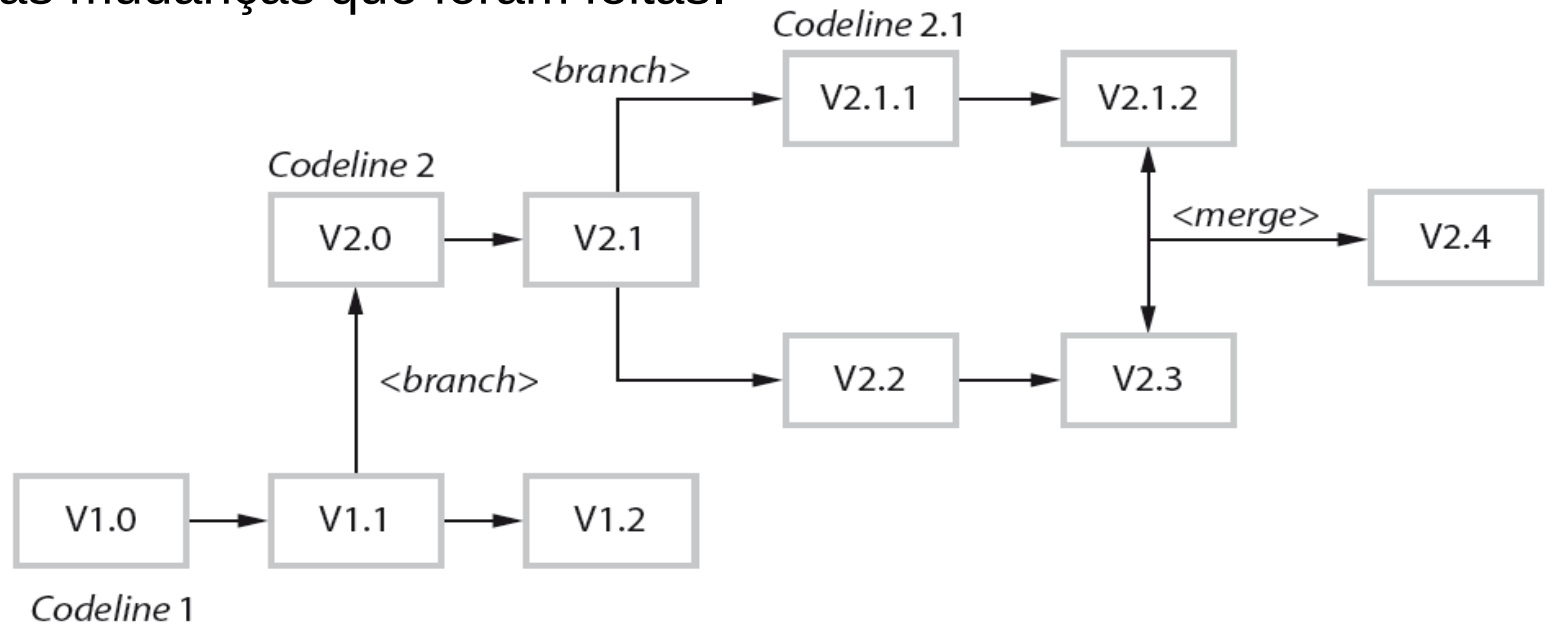


Fonte: Sommerville (2011)

Gerenciamento de Versões

- Em vez de uma sequência linear de versões das mudanças do componente ao longo do tempo, pode haver várias sequências independentes.

- Em algum momento, pode ser necessário fundir ramificações de *codelines* para criar uma nova versão do componente que inclui todas as mudanças que foram feitas.



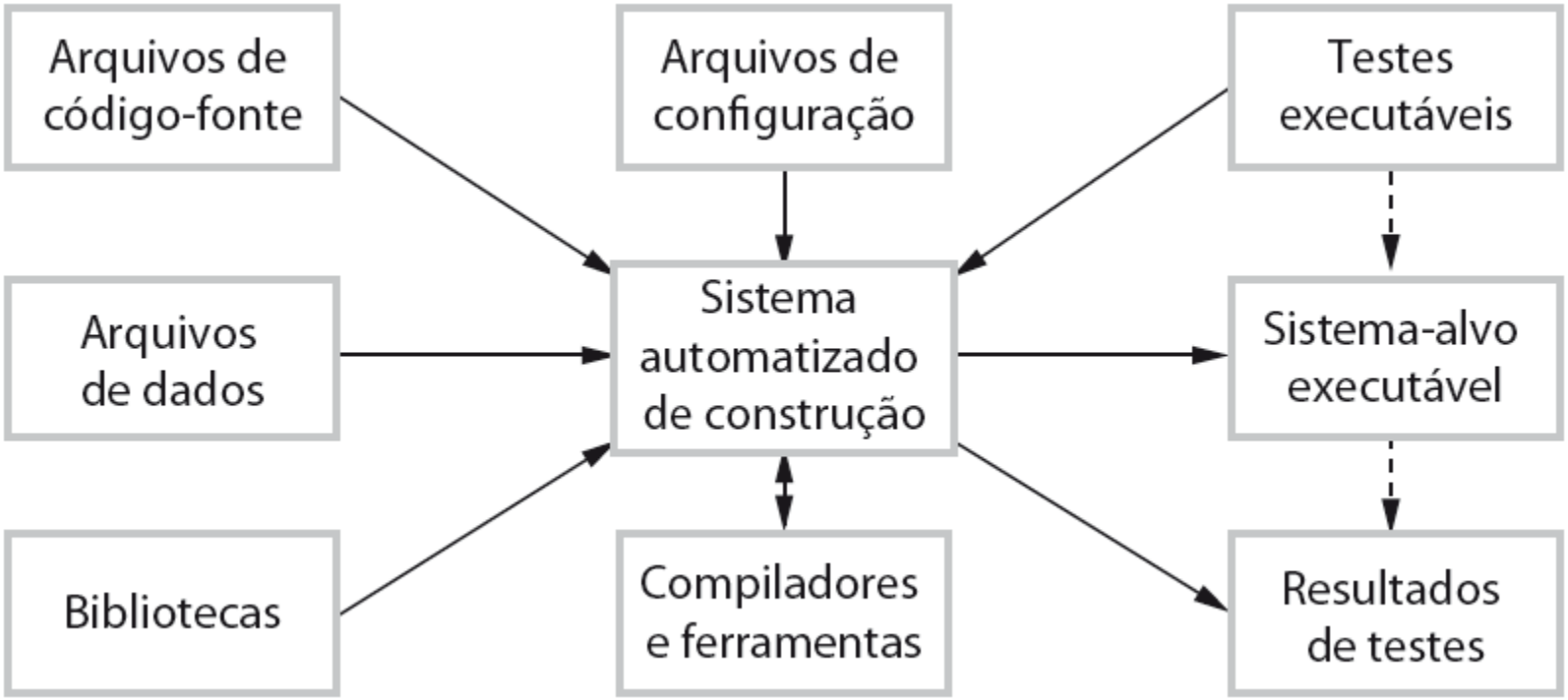
Construção de Sistemas

- É o processo de criação de um sistema completo e executável por compilar e ligar os componentes do sistema, bibliotecas externas, arquivos de configuração, etc.

- Desenvolvedores realizam *check-out* de código do sistema de gerenciamento de versões em um espaço de trabalho privado antes de fazer mudanças ao sistema

- Desenvolvedores realizam check-in de código para o sistema de gerenciamento de versões antes de ser construído.

Construção de Sistemas



Construção de Sistemas

- Geração de script de construção
- Integração de sistema de gerenciamento de versões
- Recompilação mínima
- Criação de sistemas executáveis
- Automação de testes
- Emissão de relatórios
- Geração de documentação

Recompilação Mínima

- As ferramentas de apoio à construção do sistema geralmente são projetadas para minimizar a quantidade de compilação.
- O que é feito por meio da verificação da disponibilidade de uma versão compilada de um componente. Se assim for, não existe a necessidade de recompilar esse componente.
- Uma assinatura única identifica cada arquivo e cada versão do código-objeto que é alterado quando o código-fonte é editado.
- Ao comparar as assinaturas nos arquivos de código-fonte e código-objeto, é possível decidir se o código-fonte foi usado para gerar o código-objeto do componente.

Recompilação Mínima

- ***Timestamps* de modificação**
  - ✓ A assinatura no arquivo do código-fonte é a hora e a data em que o arquivo foi modificado.
  - ✓ Se o arquivo do código-fonte de um componente foi modificado após o arquivo do código-objeto relacionado, em seguida, o sistema assume que é necessária a recompilação para criar um novo arquivo de código-objeto.

- ***Checksums* de código-fonte**
  - ✓ A assinatura no arquivo do código-fonte é uma soma calculada (um identificador único) a partir dos dados no arquivo.
  - ✓ Se você alterar o código fonte, vai gerar uma somatória diferente e então terá a certeza de que os arquivos de código fonte com diferentes somatórias são realmente diferentes.

Recompilação Mínima

- ***Timestamps***
  - ✓ Ao recompilar um componente, ele substitui o código-objeto.
  - ✓ Como os arquivos de código-fonte e código-objeto são ligados por nome, não é possível construir diferentes versões de um componente de código-fonte no mesmo diretório ao mesmo tempo.

- ***Checksums***
  - ✓ Ao recompilar um componente, não substitui o código-objeto
  - ✓ Em vez disso, gera um novo arquivo de código-objeto e *tags* com a assinatura do código-fonte.
  - ✓ É possível fazer a compilação em paralelo, e diferentes versões de um componente podem ser compiladas ao mesmo tempo.

Ferramentas para Controle de Mudanças e Versões

- Duas categorias de ferramentas:

1. **Ferramentas de SCM** que funcionam em conjunto com um repositório de dados e possuem mecanismos para identificação, versão e controle de alterações, auditoria.
   - Concurrent Version Systems (CVS)
   - SubVersion – Tortoise SVN
   - Git Hub

2. **Issue Trackers:** sistemas para gerenciar a criação, atualização e resolução de incidentes reportados por clientes ou colaboradores (**Bug ou defect trackers**).
   - Mantis Bug Tracker (BT)
   - Bugzilla

Ferramentas para Controle de Mudanças e Versões

Issue ou Bug Trackers

- Esses tipos de ferramentas servem não apenas para o controle de mudanças, mas também para gestão de requisitos, de atividades e controle de indicadores.

- O **Mantis BT** (licença GPL) é uma das mais utilizadas, testadas (testes on-line) e mantida pela comunidade.

- O **Bugzilla** foi criado e é mantido pela Mozilla, dispondo de controles de segurança, evolução e suporte.

- **Trac** (licença BSD) foi desenvolvido em Python e possui interface direta com as ferramentas de SCM.

Ferramentas para Controle de Mudanças e Versões

Ferramentas de SCM

- O objetivo principal é o armazenamento dos itens de configuração e o posterior **controle de versão** desses itens durante o processo de desenvolvimento.

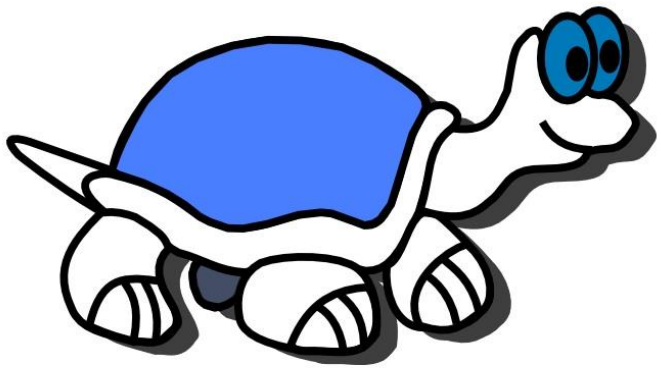
- É possível armazenar **vários tipos de itens** de configuração: códigos-fonte, formulários, scripts de banco de dados, tabelas, consultas, etc.

- Pode-se **recuperar um objeto** de uma versão anterior **ou bloquear** um objeto que está sendo usado por um dos integrantes da equipe.

- Podem ter **controle centralizado ou distribuído**.

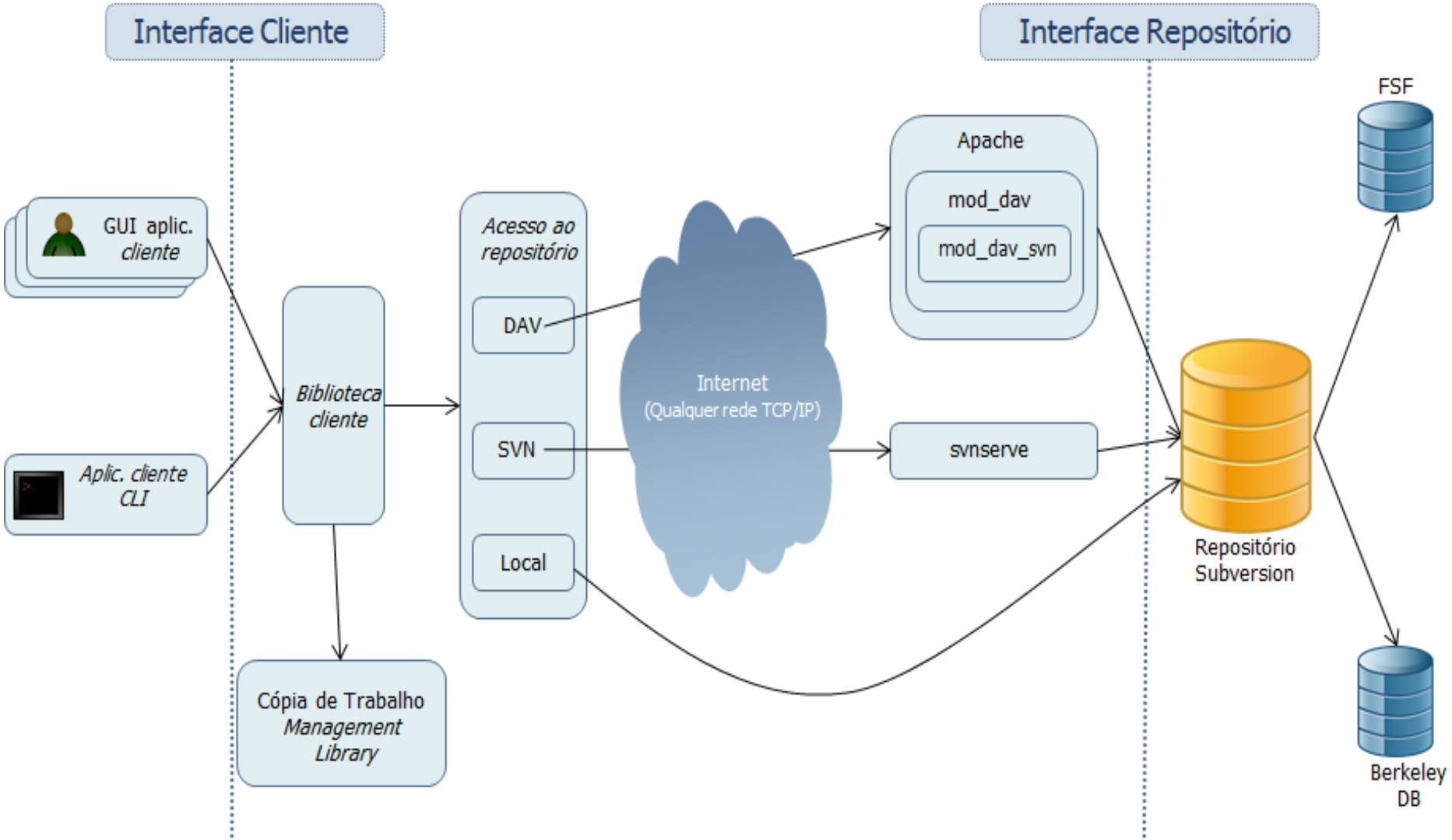
Subversion: Controle de Versão Centralizado

- Natural sucessor da ferramenta CVS, que foi a mais tradicional ferramenta de SCM por muitos anos.
- O Subversion (SVN) foi lançado em 2004 na versão 1.0 e é uma das ferramentas de controle de versão mais utilizadas no mercado há alguns anos.
- Mantido pela Apache está na versão 1.9.4, Abril 2016.
- O cliente mais usado é o Tortoise SVN.
- Integrável com várias ferramentas em diferentes SO.



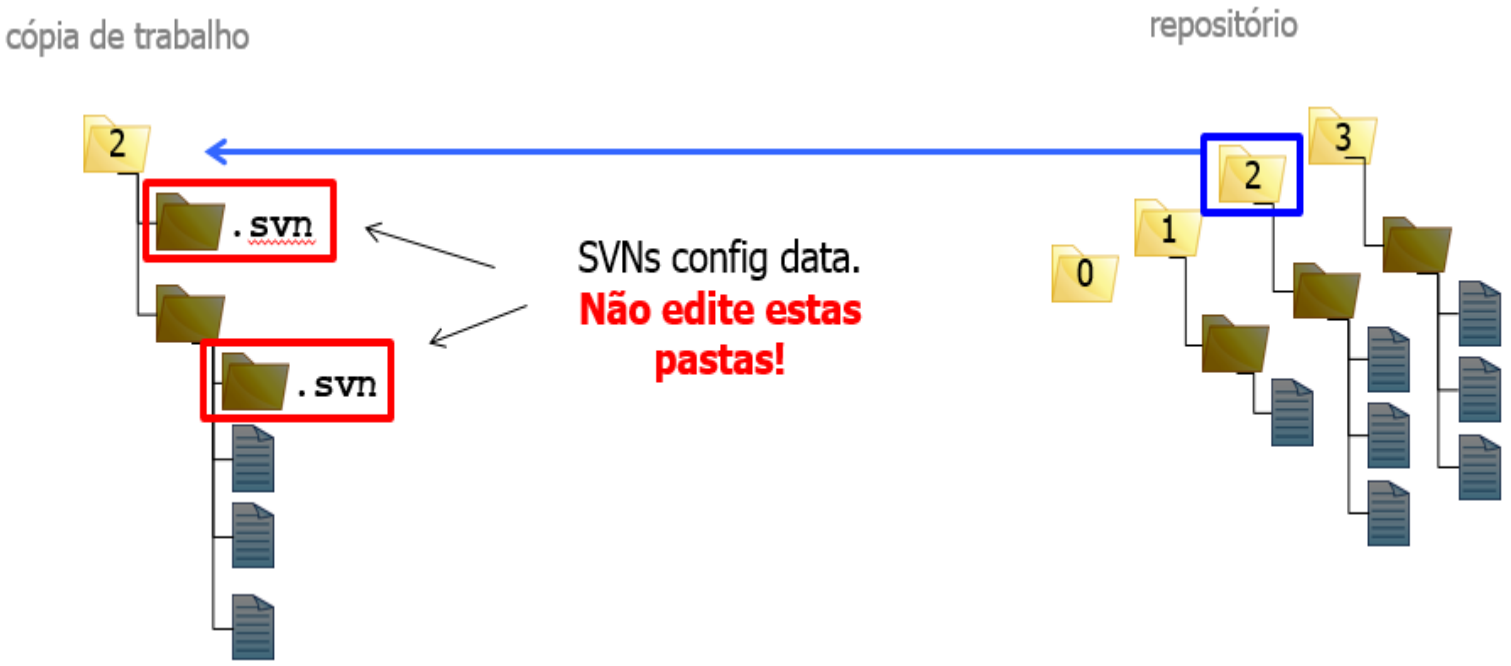
Subversion: Controle de Versão Centralizado

Arquitetura Cliente-Servidor SVN



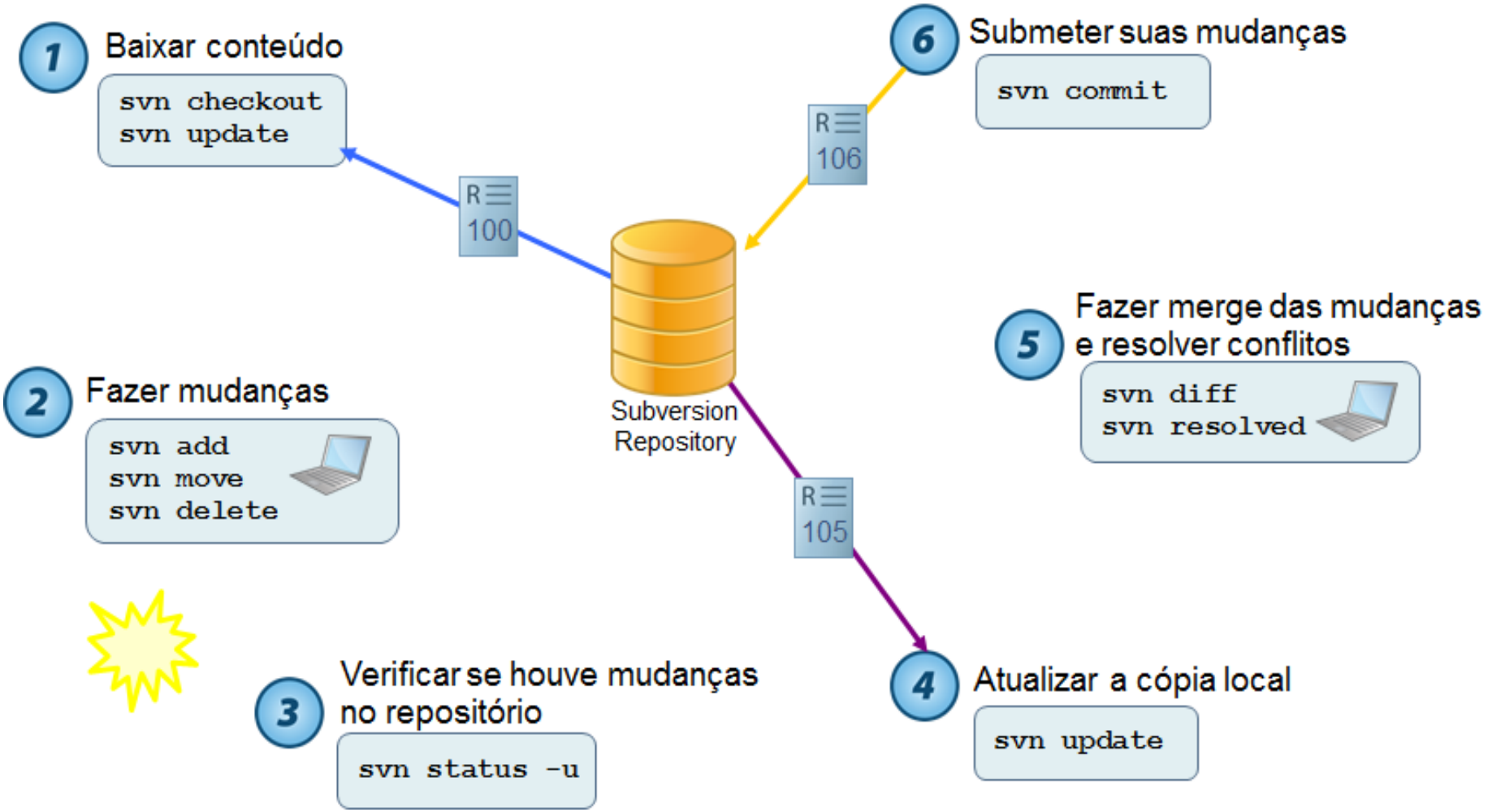
Subversion: Controle de Versão Centralizado

- O comando de Check-Out cria uma cópia de trabalho em uma pasta local e as pastas e arquivos ocultos .SVN fazem o controle de alterações e sincronismo.



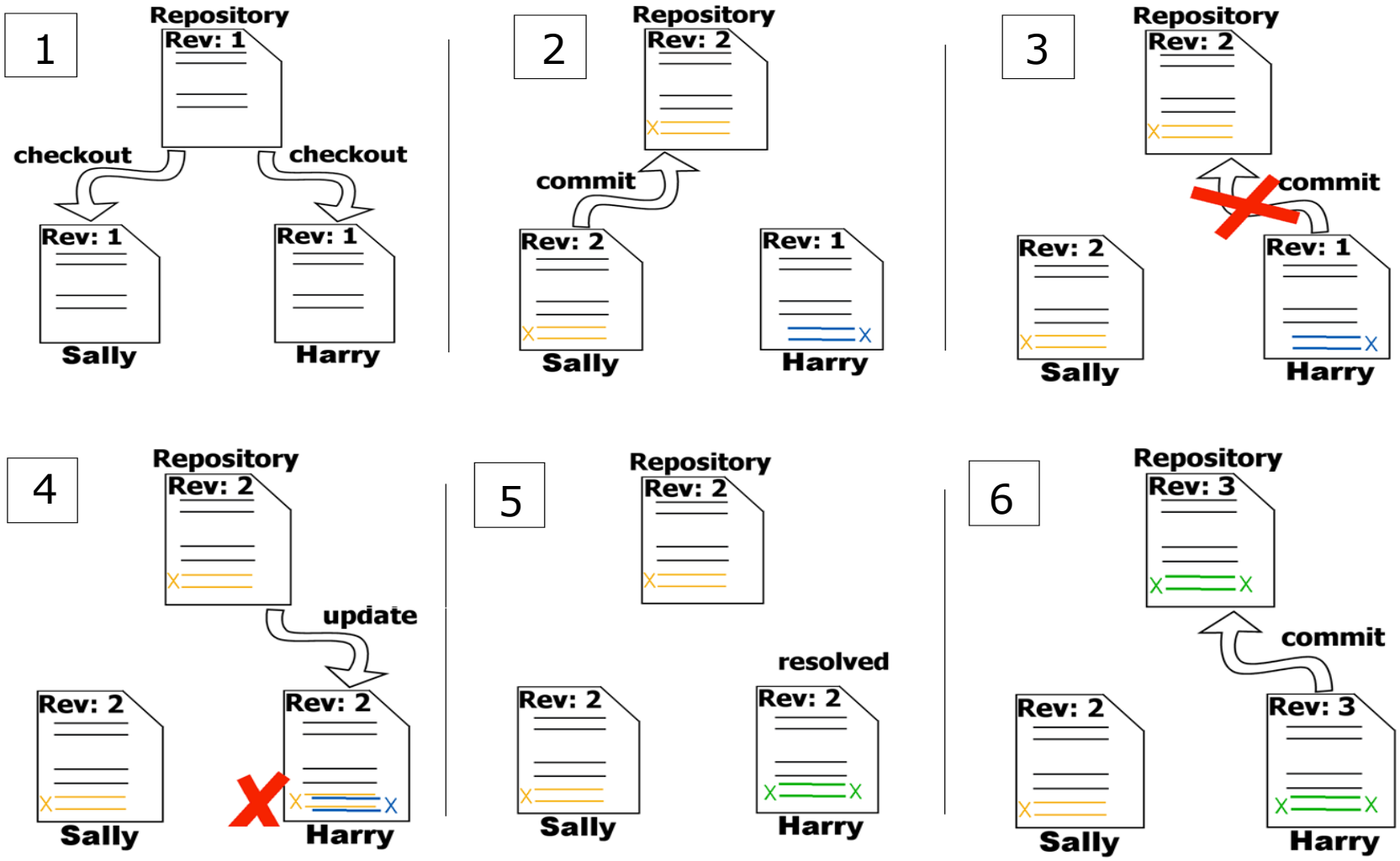
Subversion: Controle de Versão Centralizado

Configuração do Trabalho em Equipe no SVN



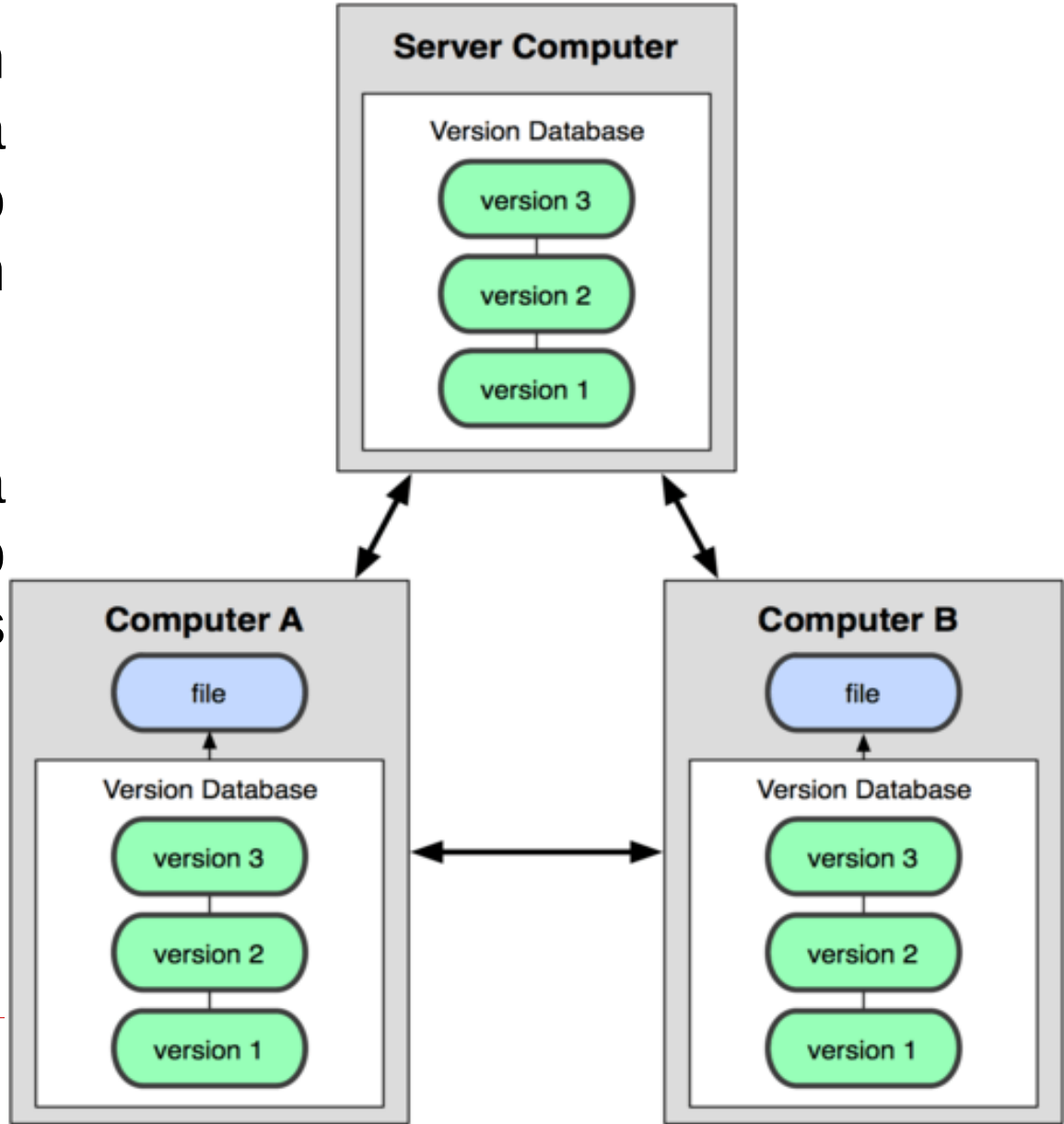
Subversion: Controle de Versão Centralizado

Controle de Sincronização



Sistemas de Controle de Versão Distribuído

- Não é necessário um servidor central já que qualquer nó funciona como um servidor.
- Cada checkout é na prática um backup completo de todos os dados do repositório.
- Git, Mercurial,
- Bazzar, Darcs.



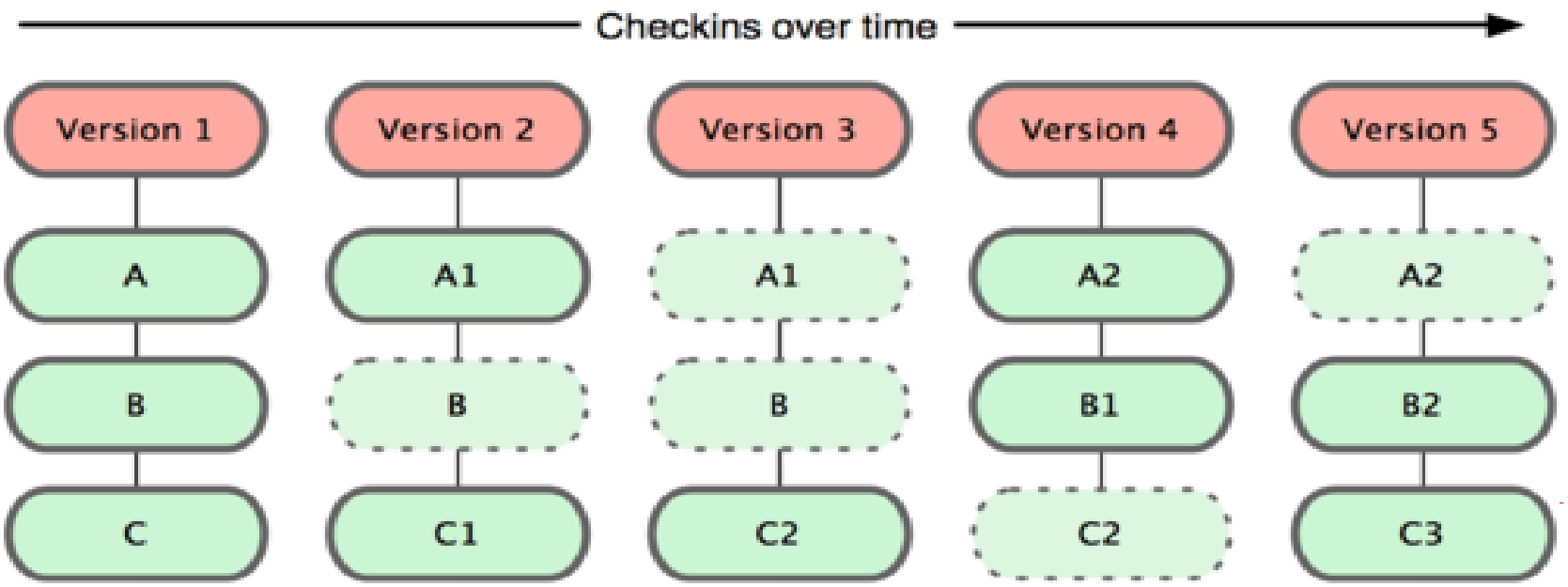
Git: Controle de Versão Distribuído

- Atualmente, a ferramenta Git é amplamente utilizada para controle de versão, sendo uma tendência corporativa.
- Utilizada a partir de 2005 pela comunidade Linux em substituição ao BitKeeper.
- Sistemas distribuídos, como o Git, permitem trabalhar com vários repositórios remotos com os quais pode-se colaborar, permitindo trabalhos em conjunto em projetos.
- GitHub é um serviço de web hosting compartilhado para projetos que controlam a versão com o Git.

Git: Controle de Versão Distribuído

- A maioria dos sistemas de controle de versão armazenam uma lista de mudanças por arquivo.

- Ao contrário, o Git considera dados como snapshots e em um commit só é armazenada uma referência, um link para arquivos idênticos aos anteriores já salvos.



Git: Controle de Versão Distribuído

- A maior parte das operações no Git precisam apenas de recursos e arquivos locais para operar.
    - Exemplo: acesso ao histórico do projeto

- Git realiza integridade dos dados através de Checksum, especificamente usando o hash SHA-1.

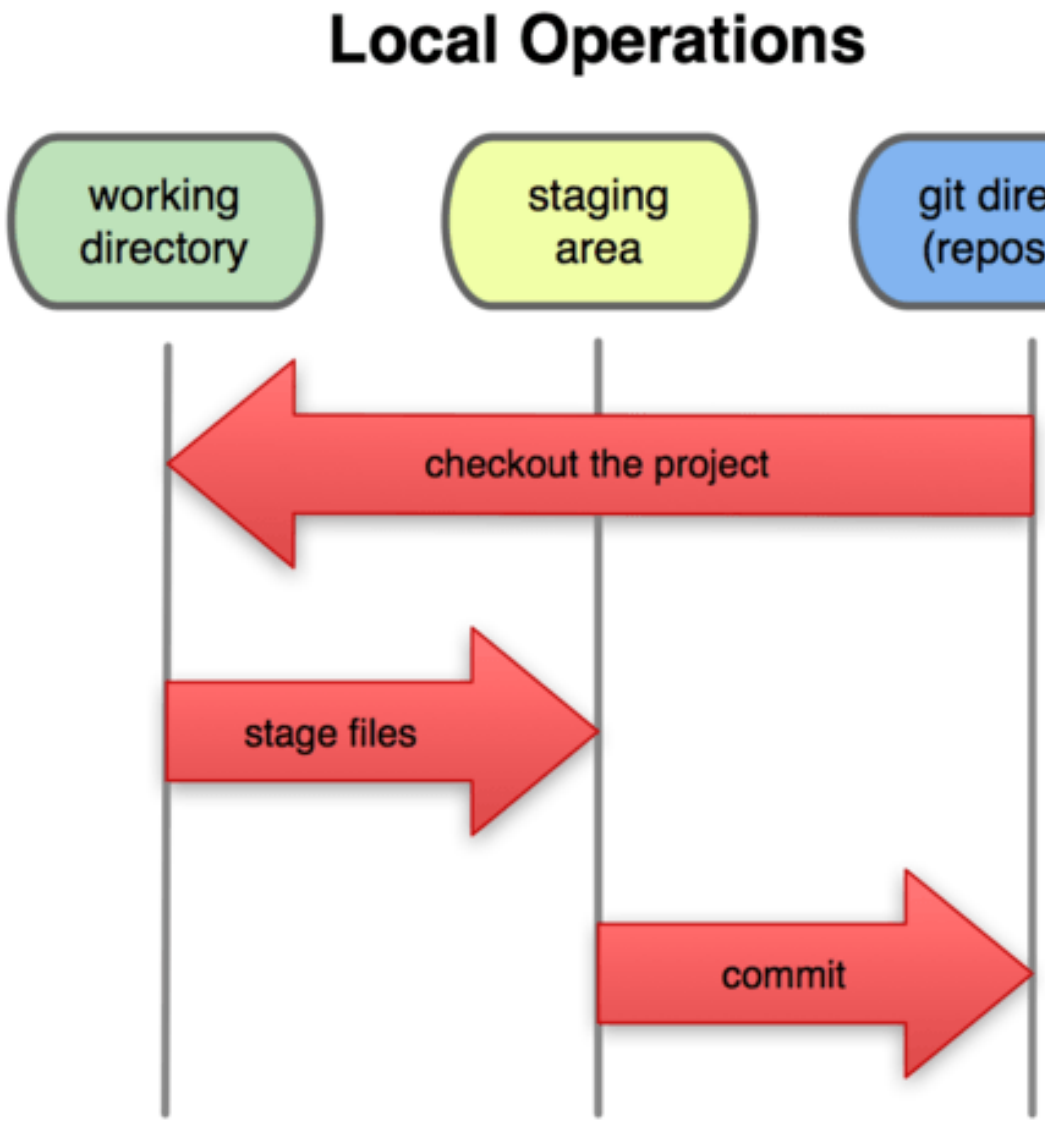
- Os arquivos controlados pelo Git estão em 3 estados:

1. **Consolidado** (commited): dados no working directory

2. **Modificado** (modified): arquivo que sofreu mudanças

3. **Preparado** (staged): arquivo é marcado como modificado para fazer parte do próximo commit.

Git: Controle de Versão Distribuído



Workflow básico do Git

1. Copie o repositório Git
2. Execute o check-out
3. Modifique arquivos no diretório de trabalho
4. Selecione arquivos, adicionando snapshots
5. Commit da área de preparação para o repositório Git

Integração Contínua

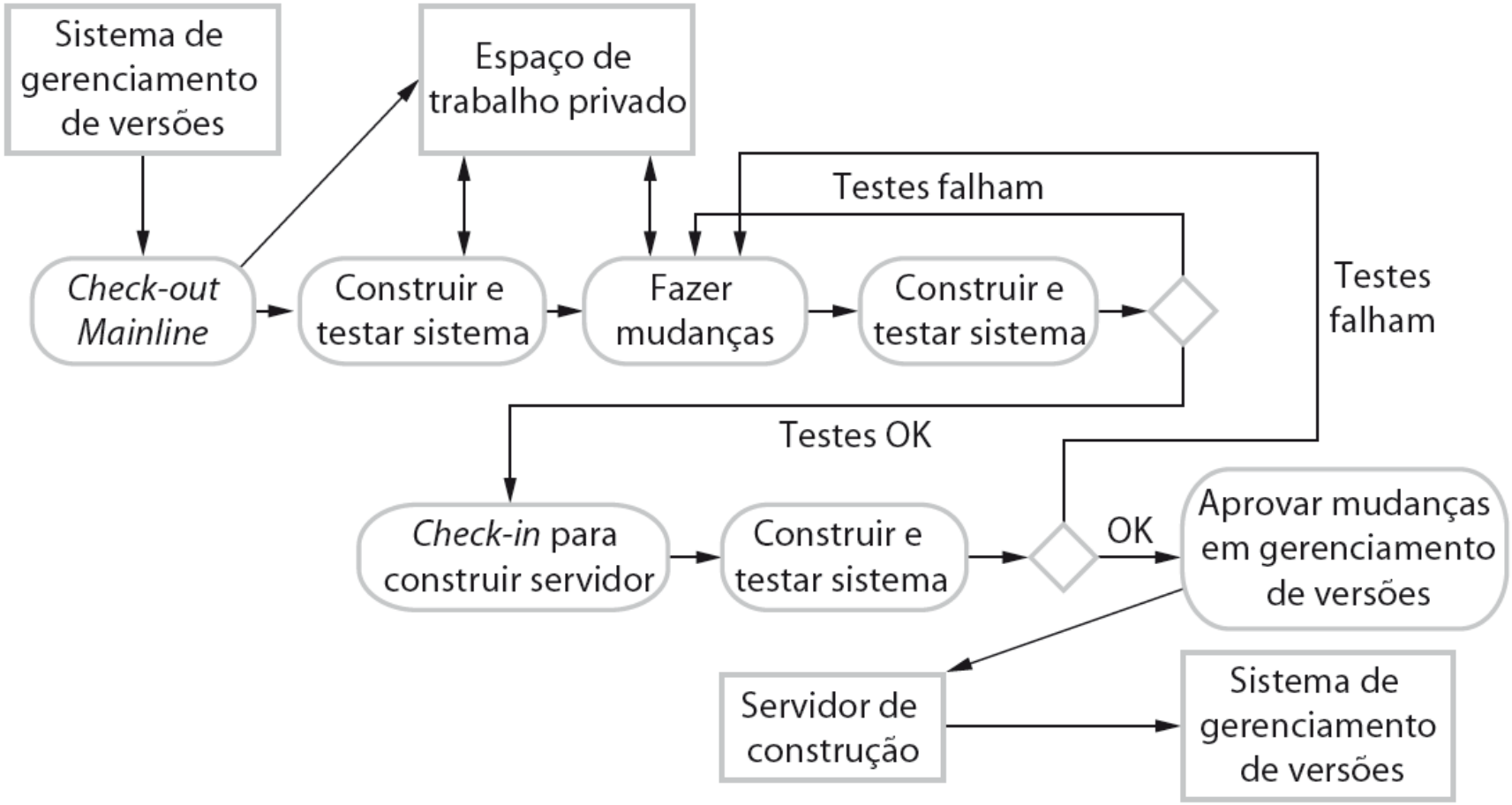
- Realizar o *check-out* do *mainline* do sistema de gerenciamento de versões no *workspace* do desenvolvedor.

- Construir o sistema e executar testes automatizados para garantir que o sistema construído passe em todos os testes.
  - Se não, a construção será interrompida e você deverá informar quem fez o último check-in do sistema de *baseline*, que será responsável pela reparação do problema.

- Fazer as mudanças para os componentes do sistema.

- Construir o sistema no *workspace* e executar novamente os testes do sistema. Se os testes falharem, continuar editando.

Integração Contínua



Integração Contínua

- Uma vez que o sistema passou nos testes, verificar no sistema construído, mas não aprovar como novo *baseline* de sistema.

- Construir o sistema no servidor de construção e executar os testes, no caso de outros componentes terem sido modificados depois de já ter acontecido o *check-out* do sistema.

- Se este for o caso, fazer o *check-out* dos componentes que falharam e editar esses testes passem em seu espaço de trabalho privado.

- Se o sistema passar nos testes sobre o sistema de construção, e as mudanças forem aprovadas, então uma nova *baseline* é adicionada a *mainline* de sistema.

Construção Diária de Versão

- A organização responsável pelo desenvolvimento define um tempo de entrega para os componentes do sistema:

  - ✓ Caso os desenvolvedores tenham novas versões dos componentes que estão escrevendo, eles devem entregá-las nesse período.

  - ✓ Uma nova versão do sistema completo é construída a partir desses componentes.

  - ✓ Em seguida esse sistema é entregue à equipe de testes, que realiza um conjunto predefinido de testes de sistema.

  - ✓ Defeitos que são descobertos durante os testes do sistema são documentados e voltam para os desenvolvedores do sistema. Eles reparam esses defeitos em uma versão posterior do componente.

Gerenciamento de Releases

- Release: versão de um software distribuído aos seus clientes.

- Normalmente existem dois tipos de release:
  - **Releases grandes** que proporcionam nova funcionalidade importante;
  - **Releases menores** que reparam bugs e solucionam os problemas dos clientes.

- Para softwares customizados, os releases do sistema podem ter que ser produzidos para cada cliente e clientes individuais podem estar executando várias versões diferentes do sistema, ao mesmo tempo.

Gerenciamento de Releases

- No caso de um problema, pode ser necessário reproduzir exatamente o software que foi entregue para um cliente particular.

- Quando é produzida uma versão do sistema, essa deve ser documentada para assegurar que possa ser recriada no futuro – fundamental para sistemas customizados e de longa vida útil.

- Os clientes podem usar um único release desses sistemas por muitos anos e podem exigir mudanças específicas para um sistema de software especial muito tempo depois da data do release original.

Documentação de Releases

- Para um documento de release, você precisa gravar as versões específicas dos componentes do código-fonte que foram usados para criar o código executável.

- Você deve manter cópias dos arquivos de código-fonte, executáveis correspondentes e todos os dados e arquivos de configuração.

- Você também deve gravar as versões do sistema operacional, bibliotecas, compiladores e outras ferramentas usadas para construir o software.

Componentes de Releases

- Além do código executável do sistema, um release também pode incluir:
  - ✓ Os arquivos de configuração que definem como o release deve ser configurado para instalações particulares;
  - ✓ Os arquivos de dados, tais como arquivos de mensagens de erro, são necessários para a operação do sistema ser bem sucedida;
  - ✓ Um programa de instalação que é usado para ajudar a instalar o sistema no hardware alvo;
  - ✓ Documentação eletrônica e em papel que descreve o sistema;
  - ✓ Empacotamento e publicidade associada que foram projetadas para esse release.

Planejamento de Releases

- Além do trabalho técnico envolvido no release, deve-se preparar material de publicidade e divulgação e estratégias de marketing para convencer os clientes a comprarem o novo release.

- **Calendário de Release**
  - ✓ Se os releases são muito frequentes ou exigem atualizações do hardware, os clientes podem não mudar para o novo release, especialmente se tiverem que pagar por isso.

  - ✓ Se os releases do sistema são muito pouco frequentes pode-se perder parte do mercado, pois os clientes mudam para sistemas alternativos.