Padrões de Gerência de Configuração

Gibeon Aquino
gibeon@dimap.ufrn.br

Material adaptado do curso do Prof. Thiago Burgos

Referência

- Todos os padrões estão contidos no livro

"Software Configuration Management Patterns: Effective Teamwork, Practical Integration"

Steve Berczuk and Brad Appleton

O QUE SÃO PADRÕES?

- Padrões são formas de representação de conhecimento, organizado de forma estruturada
- Objetiva a rápida assimilação e aplicação em um novo contexto
- Construído a partir da destilação de anos de experiência

PADRÕES EM GC

- Classificação dos padrões:

Codeline

Workspace

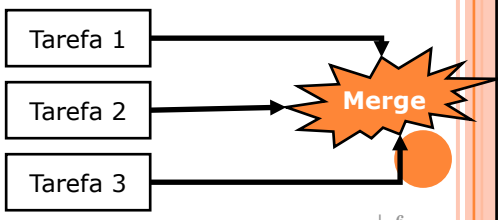
MAINLINE

Classificação: Codeline

- **Objetivo**: Simplificar sua estrutura de branches
- **Como manter várias codelines (e minimizar merging)?**

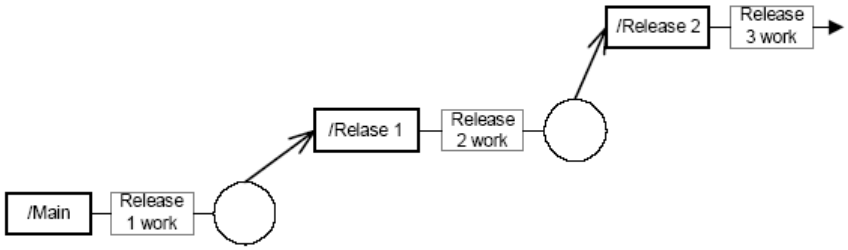
MAINLINE (CONTEXTO)

- A utilização de *Braches* é uma ótima maneira de isolar esforços paralelos.
- Exemplos de *branches* durante o desenvolvimento:
  - Variar código para plataformas
  - Manter manutenções de releases
  - Isolar esforços durante o desenvolvimento.
- Porém, isto requer *merging*, o que pode ser custoso.



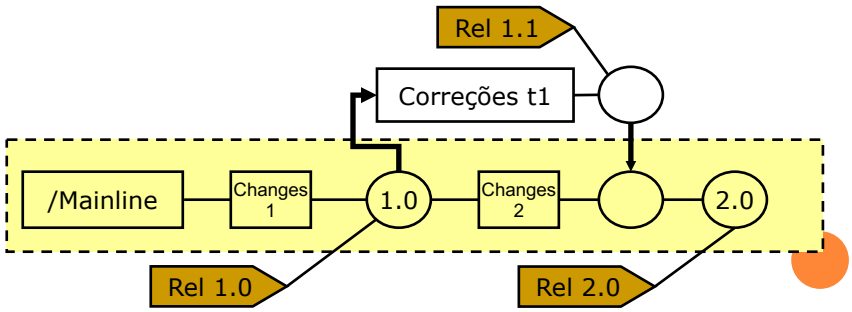
MAINLINE (CONTEXTO)

- O uso desenfreado de *branches* pode gerar estruturas complexas, difíceis de serem mantidas e integradas.



MAINLINE (SOLUÇÃO)

- **Controle o uso de *branch* elegendo uma linha de desenvolvimento principal** que
  - Agrega todos os esforços
  - Serve de base para outras *codelines*
  - Reduz custo com *merging*



Active Development Line

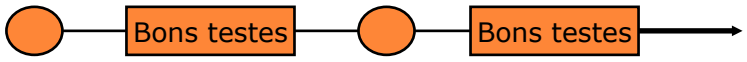
- *Classificação: Codeline*
- Objetivo: Desenvolver uma mainline estável e de rápida evolução.
- **Como manter a mainline estável para que seja útil?**

Active Development Line (Contexto)

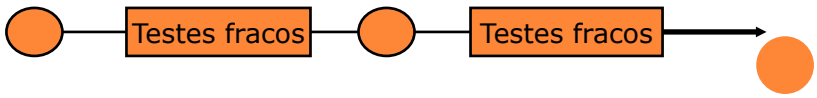
- A mainline é um ponto de sincronização
  - Requer a comunicação entre os desenvolvedores
  - Check-ins e integrações freqüentes são bons
  - Check-ins com má qualidade afeta a todos
- Procedimentos rigorosos para atividades
  - Demoram muito tempo
  - Geram menos check-ins
  - Logo, atrapalham o projeto

Active Development Line (Contexto)

- Cenários de Uso da Mainline
  - Mainline **pouco ativa e muito estável**

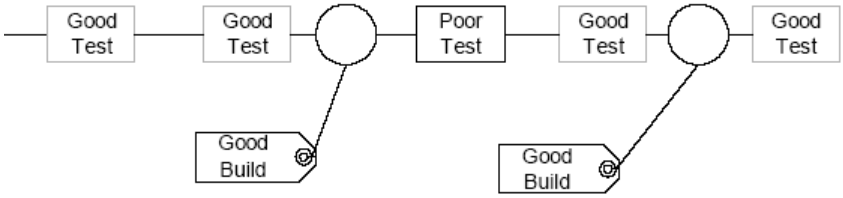


  - Mainline **muito ativa e pouco estável**



Active Development Line (Solução)

- **Use uma linha de desenvolvimento ativa**
  - Com políticas para a estabilidade necessária da codeline para suas necessidades
  - Considerando, o próprio **Ritmo** de desenvolvimento.



Private Workspace

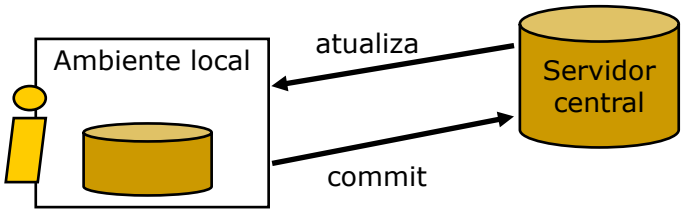
- *Classificação:* Workspace
- Objetivo: Manter um ambiente isolado de desenvolvimento.
- **Como evoluir sem se distrair com as freqüentes mudanças do ambiente?**

Private Workspace (Contexto)

- Atualização freqüente do ambiente evita o trabalho com artefatos desatualizados
  - Porém, pessoas não conseguem raciocinar em um ambiente em constante mudança
- Por outro lado, Isolamento em excesso é proibitivo, e termina sendo fator complicador.

Private Workspace (Solução)

- **Crie um ambiente de trabalho privado que contenha tudo o que você necessita** para executar suas tarefas em uma codeline
  - Você controla quando atualizá-lo
- Antes de integrar suas tarefas
  - Update
  - Build
  - Test



Private Workspace

- Pode conter:
  - Código-fonte a ser editado
  - Componentes compilados localmente
  - Documentação do projeto
  - *Scripts* de *build*
- Não pode conter:
  - Artefatos que não estão no repositório
  - Componentes que estão no repositório, mas que você copiou de um outro local desconhecido

REPOSITORY

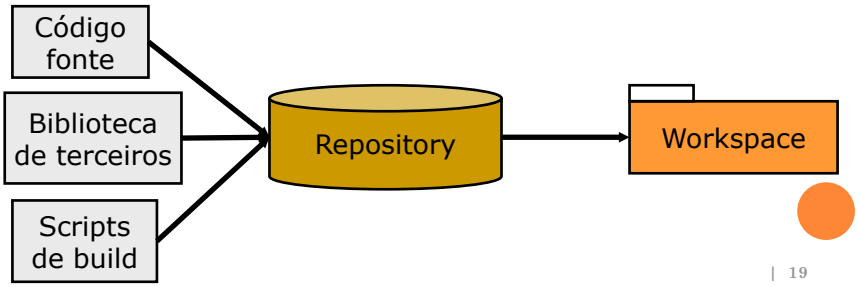
- *Classificação:* Workspace
- Objetivo: manter os itens em um local centralizado.
- **Como obter a versão correta de um determinado componente?**

REPOSITORY (CONTEXTO)

- Diferentes artefatos compõem um workspace
  - Codigo, scripts, componentes, bibliotecas
- **Integradores**, **Testadores** e **Desenvolvedores** precisam, quando necessário, ter acesso às mesmas versões do sistema

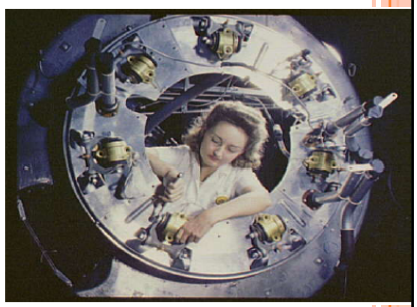
Repository (Solução)

- **Ponto único de acesso a todos os artefatos** em suas devidas codelines
  - Mecanismos de acesso
  - Replicação de workspaces
  - Organização de todas as versões de cada itens de configuração



Private System Build

- *Classificação:* Workspace
- Objetivo: Compilar e testar o que está em seu *Private Workspace*
- **Como verificar que suas mudanças não quebram o sistema antes de realizar um *check in*?**



Private System Build (Contexto)

- Cada build individual, coopera para o funcionamento do build geral.
  - O *build* do sistema pode ser complicado
  - É necessário manter consistência com outros builds
- Evitar *commits* de mudanças que interropam o sistema
  - Atenuar isolamento de um Private Workspace

Private System Build (Solução)

- Construa o sistema **individualmente** usando os mesmos mecanismos do build geral de integração
  - Use os mesmos componentes e ferramentas
  - Inclua todas as dependências requeridas
- Faça isto **antes do checkin** dos seus trabalhos
  - Inclua chamada para testes durante o build

Integration Build

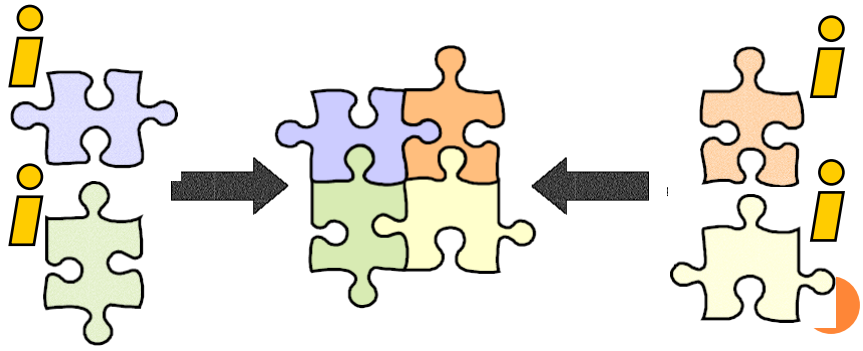
- *Classificação:* Workspace
- Objetivo: O trabalho de cada workspace precisa ser centralizado
- **Como garantir que o sistema funciona quando juntar todas as partes ?**

Integration Build (Contexto)

- Trabalhos individuais (Private workspaces) devem ser **integrados**
- Você quer garantir que o que é atualizado no repositório funciona em conjunto
- Private System Build é uma forma de testar a compilação do sistema, mas somente com as alterações que são feitas individualmente (Private workspaces).

Integration Build (Solução)

- Execute um build centralizado para todo o sistema em um ambiente isolado



Integration Build (Processo)

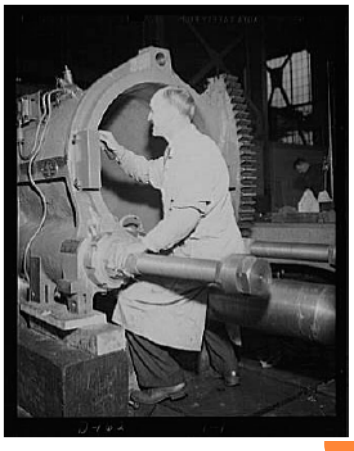
- Determine a **freqüências** de acordo com
  - O tempo de duração do build
  - A freqüência das mudanças
- O processo de build deve ser:
  - **Reprodutível**
  - Tão próximo quanto possível de um **build de produto final**
  - Automatizado, com mínima intervenção manual
  - Um mecanismo de *log* e de notificação (e-mail) para sanar problemas rapidamente

Integration Build

- Identifique builds com uma tag
  - Build geralmente é um candidato a uma baseline.
- Se um build falhar continuamente, adicione mais verificações antes dos commits.

Task Level Commit

- *Classificação:* Workspace
- Objetivo: Realizar commits por tarefas.
- **Quanto você deve trabalhar antes de realizar um commit no Repositório?**



Task Level Commit (Contexto)

- É tentador realizar várias pequenas mudanças de uma única vez
- Quanto menores as mudanças associadas a um check-in,
  - Mais fácil é voltar atrás.
  - Mais fácil é identificar problemas

Task Level Commit (Solução)

- Efetue um commit por tarefa consistente (ou sub-tarefa, com granularidade adequada)
- Considere a complexidade de cada tarefa, se pode ser subdividida, etc.
- Se houver dúvida, induza o erro para a granularidade menor
  - Isto aumenta a freqüência dos commits
  - Aumenta a possibilidade de desfazer, de forma segura, o trabalho feito

Task Level Commit

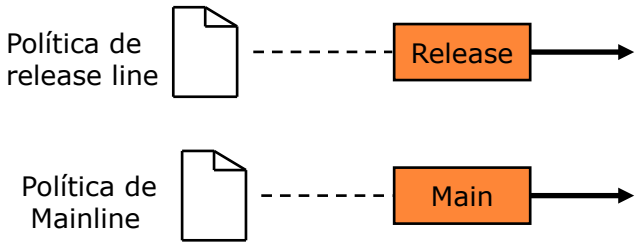
- Algumas modificações são intrinsecamente longas
  - Utilize **Task Branch**, e repita as considerações propostas por **Task Level Commit** dentro do novo branch
- Antes de realizar um *commit*, assegure que seu *workspace* está com as versões mais novas.
- Utilize testes e private system builds para assegurar mudanças consistentes

Codeline Policy

- *Classificação:* Codeline
- Objetivo: Ditar regras para as codelines.
- **Como saber a forma de proceder e de trabalhar de cada codeline?**

CODELINE POLICY (CONTEXTO)

- Diferentes *codelines* tem diferentes requisitos de estabilidade
- Como explicar uma política?
  - Qual o nível de documentação que você precisa?
  - Como motivar as pessoas a utilizá-la?



CODELINE POLICY (SOLUÇÃO)

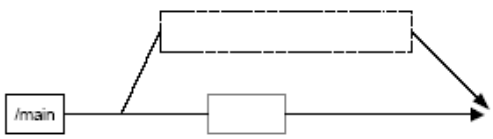
- Defina **regras** para cada codeline como uma Política do Codeline, que
  - Determine como e quando as pessoas realizaram mudanças.
  - Seja concisa e passível de auditoria.
- As políticas de cada *codeline podem* conter:
  - **Tipo de trabalho** sugerido (desenvolvimento, release, etc)
  - **Como e quando** novos elementos podem sofrer *check-in*, *check-out*, *branch* e *merge*
  - **Restrições de acesso** para indivíduos, papéis e grupos

Task Branch

- *Classificação:* Codeline
- Objetivo: Não "sujar" a linha principal de desenvolvimento com modificações de longa duração ?
- **Como realizar trabalhos de longa duração sem atrapalhar a mainline?**

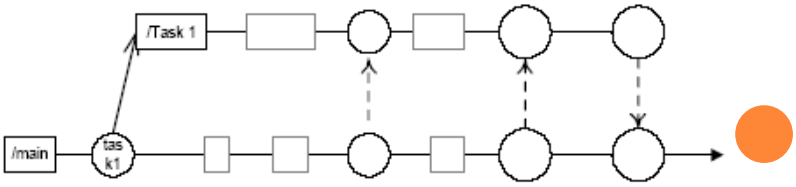
Task Branch (Contexto)

- Às vezes somente parte da equipe está trabalhando em uma tarefa
  - A tarefa pode incluir vários passos, e ser longa
- Exemplo:
  - Grandes *refactorings* tomam tempo e geram instabilidade temporaria.
    Ex.: *novo mecanismo de persistência*



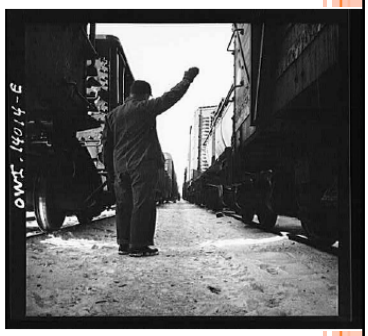
TASK BRANCH (SOLUÇÃO)

- Use *Branches* para isolar a *Mainline*
  - Crie um *branch* em separado para cada atividade que tiver mudanças significantes para a *codeline*.
  - Use como um mecanismos para reduzir riscos
- Para facilitar o *merge final do branch, i*ntegre constantemente mudanças na codeline (Up-merge)



RELEASE LINE

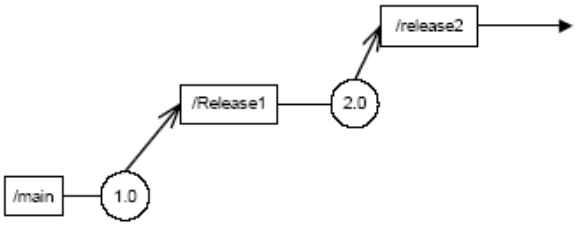
- *Classificação:* Codeline
- Objetivo: Separar a codeline do release, da continuação do desenvolvimento.
- **Como manter uma versão de release sem interferir no trabalho corrente?**



Release Line (Contexto)

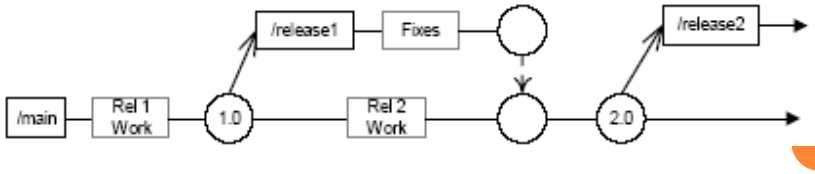
- Nem tudo pode ser evoluído na *mainline*
  - Release precisa de uma política com mais estabilidade e sem interferir no desenvolvimento

- Precisa-se evitar estruturas complexas



Release Line (Solução)

- Separe **manutenção/release** e **desenvolvimento ativo** em branches separados.
- Mantenha cada *release* em uma *release line* independente para possibilitar **futura a correção de bugs.**

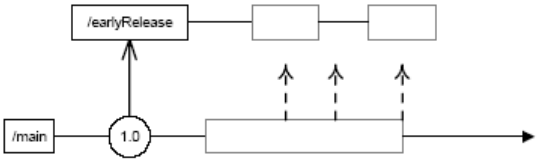


Release-Prep Codeline

- *Classificação:* Codeline
- Objetivo: Estabilizar o código do release separando do desenvolvimento regular.
- **Como estabilizar uma codeline para um release, sem parar o trabalho na mainline?**

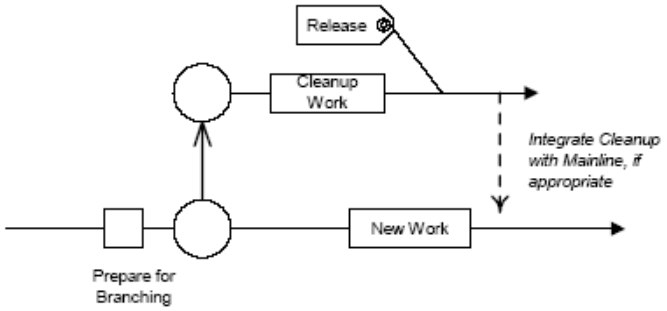
Release-Prep Codeline (Contexto)

- A **Mainline** precisa ser estabilizada para que o release seja terminado
  - Congelar novos trabalhos na **mainline** é penoso
- Uma Release-prep codeline é simplesmente a antecipação da **release line** para antes do momento do release.

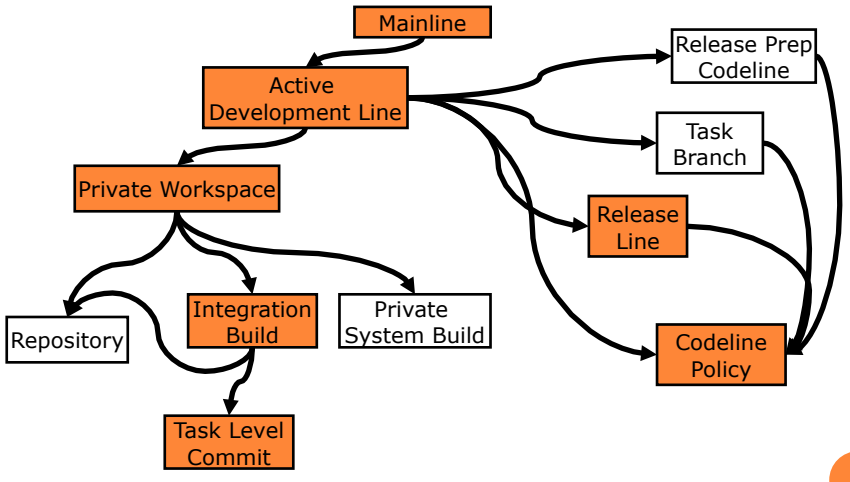


Release-Prep Codeline (Solução)

- Crie um novo branch *(**Release-Prep Codeline**)* quando o sistema estiver perto se da qualidade desejada do release.
  - Finalize o *release* neste *branch*, e torne-o o ***Release Line***



Relação entre os padrões

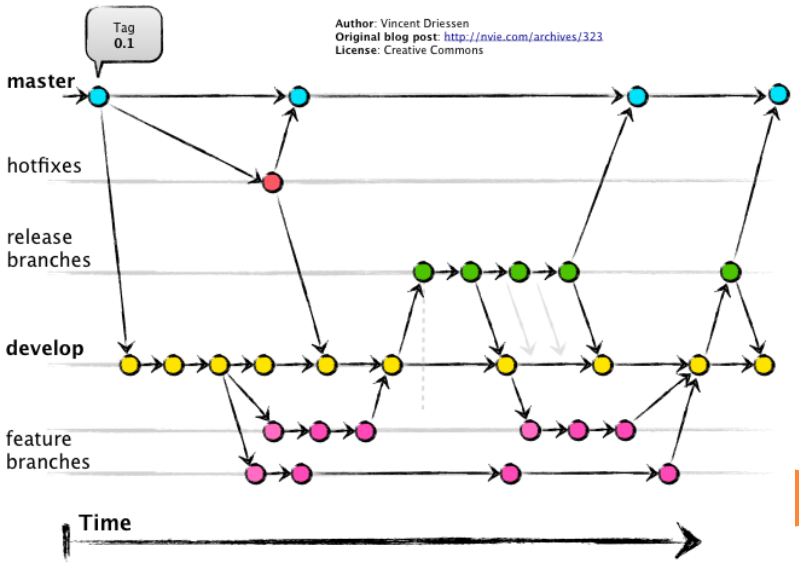


[Berczuk, S., Appleton, B., Software Configuration Management Patterns]

GITFLOW

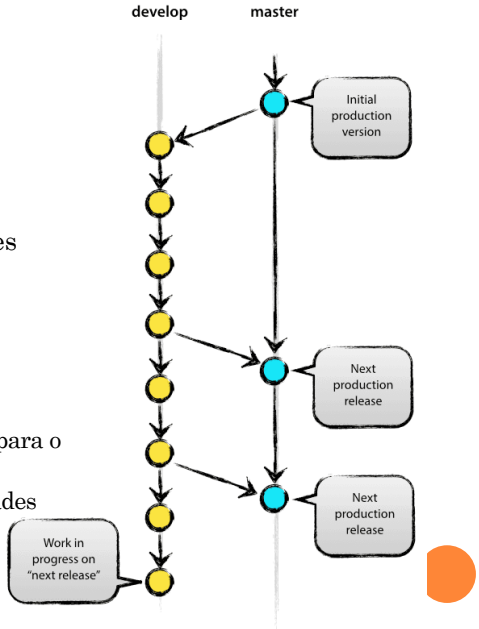
- Um modelo para organização do trabalho de evolução e desenvolvimento de projeto
- Padrão para gerenciamento de versões em um projeto
  - Convenciona os nomes das branches
  - Estabelece políticas de evolução para commits e merges nas branches
- Uma forma organizada de permitir o trabalho colaborativo em um projeto
- Instância vários padrões e boas práticas de Gerência de Configuração

FLUXO



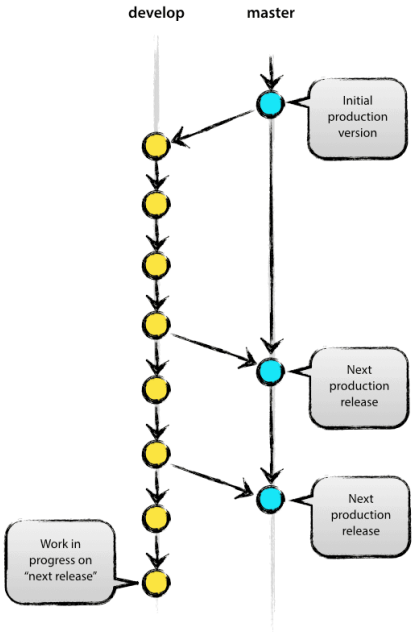
BRANCHES MASTER E DEVELOP

- Branches principais
- **Master** é usada para releases (sempre estável)
  - Armazena apenas versões de produção
- **Develop**
  - Armazena código instável
  - Principal branch de trabalho para o time
  - Contém as novas funcionalidades



REGRAS PARA BRANCHES MASTER E DEVELOP

- Próximo ao release:
  - A estabilidade da branch **develop** é testada
- Quando o código do branch **develop** é considerado estável:
  - As alterações são mescladas de volta para o branch **master**
  - Cria-se uma tag na **master**
- Continua o desenvolvimento da próxima release na **develop**



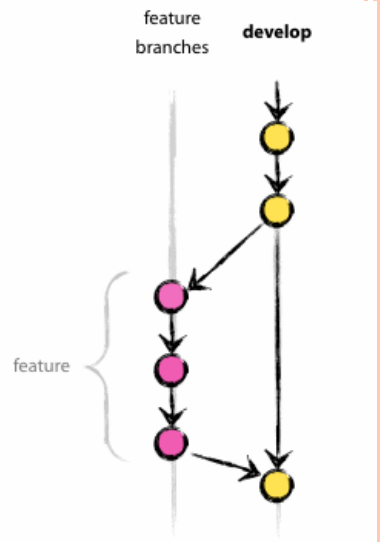
Branches de suporte

- Branches de melhorias (feature)
- Branches de lançamento (release)
- Branches de correções (hotfix)

Cada tipo de branch tem um propósito específico e segue regras de quais branches devem ser originados e mesclados

Feature Branches

- Usadas para desenvolver novas funcionalidades para a próximo release
- Gerada a partir da branch **develop**
- Tempo de vida curto (apenas durante o desenvolvimento da melhoria)

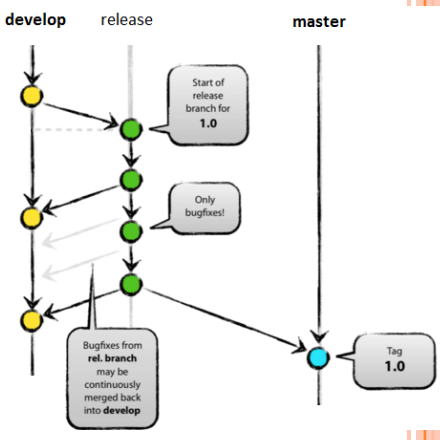


Regras para Features Branches

- Deve ser mesclada ao branch **develop** quando a melhoria for concluída
- Deve ser eventualmente atualizada a partir do código da **develop**
- São candidatas a serão removidas, após o merge com a **develop**

Release Branches

- Usadas para preparação do lançamento da próxima versão de produção
- Criadas a partir da **develop**
- Correção de bugs para estabilização da versão
- O desenvolvimento de novas funcionalidades continua em paralelo na **develop**

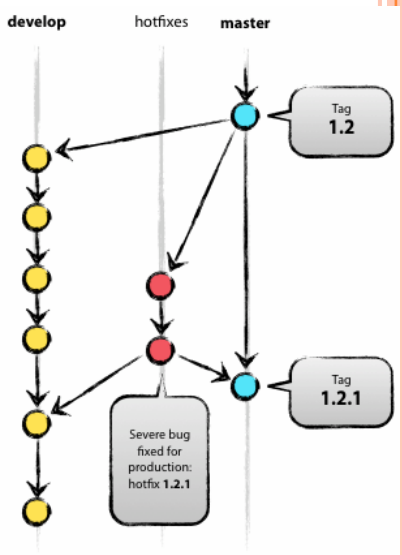


Regras da branch de release

- Nenhuma nova funcionalidade pode ser adicionada
- Mesclada de volta com a **master**, quando estiver suficientemente estável para ir para produção
  - Pode ser removida, após ser mesclada
- Deve-se mesclar as correções à **develop** quando finalizadas
- Convenção de nomes: ***release-****
- Apenas uma deve existir por vez

Hotfix Branches

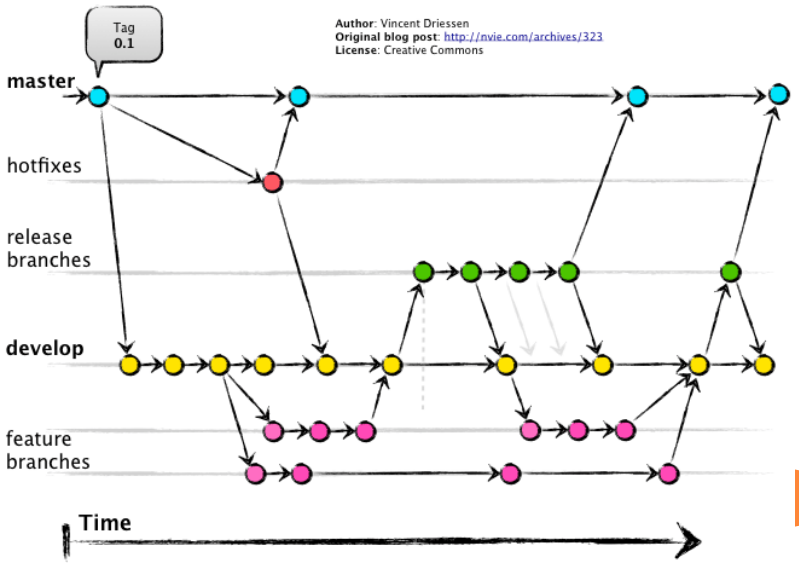
- São como as branches de release, mas para correção de bugs
- Criadas a partir da **master**
- Devem ser mescladas de volta a **develop** e **master**



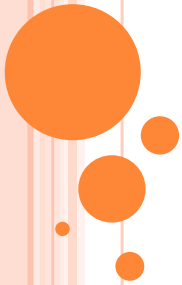
Regras da branch de hotfix

- Enquanto algúem corrige o problema o time continua o trabalho em **develop**
- Convenção de nomes: ***hotfix-****
- Pode ser removida, após ser mesclada a **master** e **develop (**e talvez **release**)

Fluxo



Padrões de Gerência de Configuração



Gibeon Aquino
gibeon@dimap.ufrn.br

Material adaptado do curso do Prof. Thiago Burgos