Streamed Lines: Using Patterns for Parallel Software Development

Brad Appleton, Stephen P. Berczuk, Ralph Cabrera, Robert Orenstein

1998

Apresentado por:
Gabriel Piton Tessarolli
Wallace da Silva Ribeiro

Agenda

- Introdução
- Os padrões
  - Elementos básicos
  - Políticas de ramificação
  - Criação de ramos
  - Estruturação de ramos
- Como utilizar
- Considerações finais

Introdução

- Desenvolvimento paralelo
  - Reduzir *time-to-market*
  - Manter diversas *releases*
  - Suportar diversas plataformas
  - Organizar o processo (vários papéis)
- Problemas
  - Divisão do trabalho
  - Concorrência
  - Propagação de mudanças
  - Retrabalho

Introdução

- Motivação
  - Criar e documentar padrões aplicáveis a várias situações de desenvolvimento de software

Introdução

- Forças da ramificação e do desenvolvimento paralelo
  - Segurança
  - Permitir a evolução (*liveness*)
  - Reusabilidade
  - Trabalho em equipe
  - Suporte de ferramentas

Elementos Básicos:
Ramos e Linhas

E1. Ramo de Atividade

- Ramo utilizado para uma unidade de esforço discreta e logicamente atômica

- Pode ser uma correção, nova funcionalidade ou construção e liberação

E2. Ramo Funcional

- Ramo utilizado para uma unidade de funcionalidade lógica discreta.
  - Ramo de funcionalidade
  - Ramo de correção

- Enfoque na funcionalidade

E3. Ramo de Componente

- Ramo utilizado para se trabalhar em um específico componente do sistema

E4. Ramo de projeto

- Ramo utilizado para representar uma linha separada de desenvolvimento do projeto ou de um subprojeto.

E5. Linha de Desenvolvimento

- Uma *codeline* usada para novas funcionalidades ou manutenção

E6. Linha de Manutenção

- *Codeline* utilizada para manutenção (correção de bugs e pequenas melhorias).

E7. Linha de Integração

- *Codeline* utilizada para se realizar junções

E8. Linha de *Release*

- *Codeline* que representa o agrupamento lógico das funcionalidades entregues (release ou patch).

Padrões de Política de de Ramificação

P1. Política de *codeline*

- Contexto
  - Desenvolvimento em sistema que possui múltiplos *codelines*
- Problema
  - Qual *codeline* utilizar? Quando salvar o trabalho?
- Forças
  - Cada *codeline* tem uma “razão de existir”
  - O nome da *codeline* não consegue exprimir o significado completo
  - Problemas caso seja utilizada a *codeline* errada
- Solução
  - Ramos (*codelines*) com nomes mnemônicos
  - Descrição da “razão de existir” da *codeline*
  - Restrições de acesso
  - Divulgação da política (se possível, diretamente na ferramenta)

P2. Propriedade (*ownership*) da *codeline*

- Contexto
  - Múltiplos *codelines* com políticas definidas
  - Tarefa não coberta ou detalhada pela política definida
- Problema
  - A tarefa deve ser realizada ou não? Como decidir, garantindo a integridade e consistência da *codeline*?
- Forças
  - A política da *codeline* nem sempre consegue cobrir todos os pontos
  - Violação da política da *codeline*
- Solução
  - Associar um “dono” a cada *codeline*
    - Esclarecer pontos da política, caso necessário
    - Decidir em casos não cobertos
    - Tomar ações em caso de violação
  - Não implica em acesso exclusivo
- Variantes
  - P2.1 Ditadura da *codeline*
    - Mais restritiva

P3. Linha com acesso permissivo

- Contexto
  - É necessário determinar uma política de controle de acesso da *codeline*
- Problema
  - Quão restritiva ou exclusiva deve ser essa política?
- Forças
  - Quantidade de desenvolvedores e a experiência deles
  - Nível de risco e complexidade
  - Garantia da integridade e consistência
- Solução
  - Utilitar uma política mais permissiva:
    - *Codelines* de desenvolvimento e manutenção
    - Grupos pequenos de desenvolvedores
  - Garantir a ciência do dono da *codeline*
- Variantes
  - P3.1 Acesso com junção controlada
    - Bloqueia junção com conflito

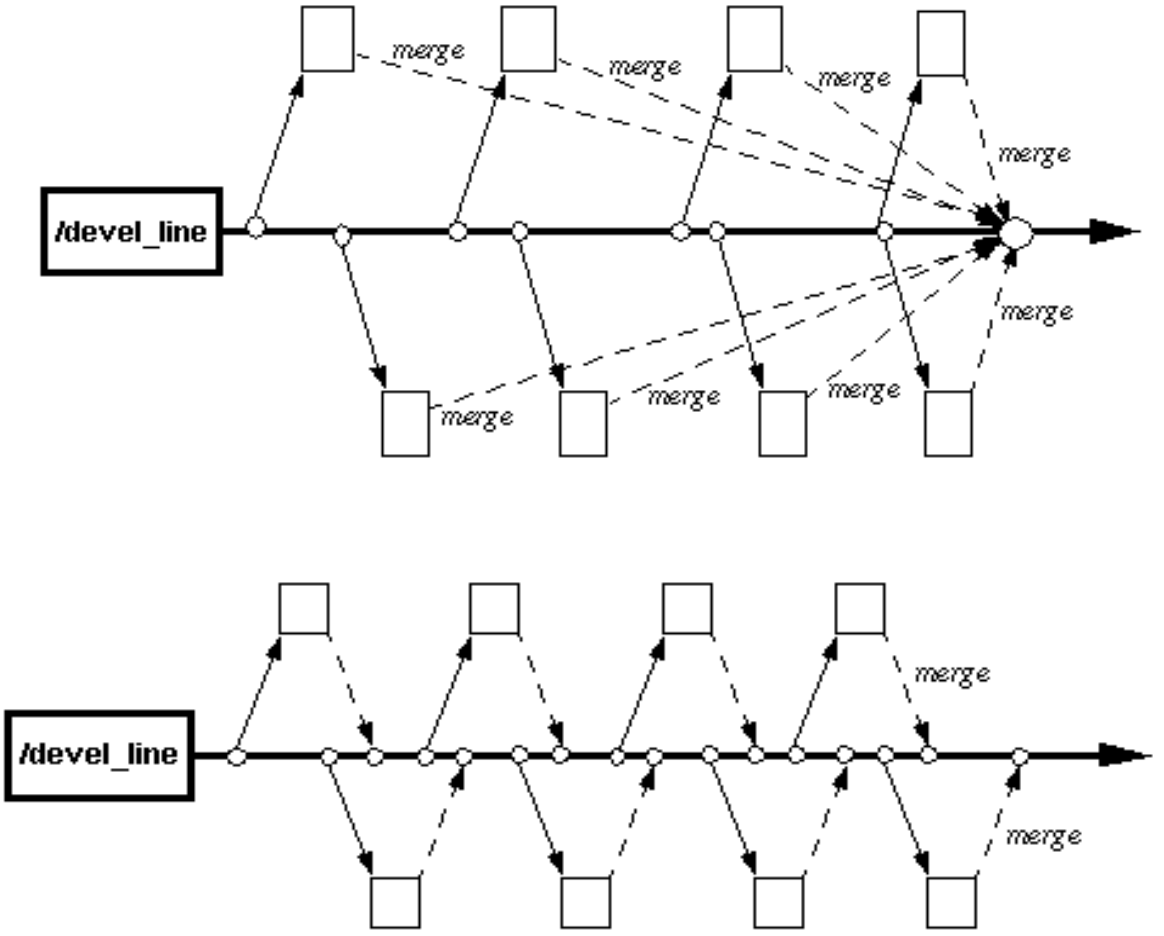
P4. Linha com acesso restrito

- Contexto
  - Idem P3
- Problema
  - Idem P3
- Forças
  - Idem P3
- Solução
  - Utilizar uma política mais restritiva
    - Grande número de desenvolvedores ou grande parte inexperiente
    - Ramos de integração
    - Fases mais avançadas do projeto
- Variantes
  - P4.1 Congelamento de *codeline*
  - P4.2 Bloqueio de exportação
    - Trava somente no momento do *checkin*
  - P4.3 Linha Privada
  - P4.4 Acesso baseado em papéis

P5. Fazer junção antecipada e frequente

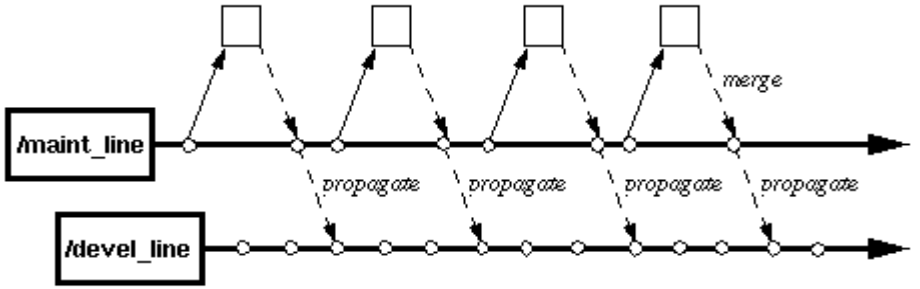
- Contexto
  - Várias fontes de novas versões para serem integradas
    - Trabalhos diretos na *codeline*
    - Ramos de tarefas
    - Linhas de subprojeto
    - Linhas remotas
- Problema
  - Quando fazer a junção? Quão frequente realizar integração?
- Forças
  - A frequência afeta a estabilidade
  - O gerenciamento de risco orienta a mitigar os problemas (junções) o quanto antes
  - A complexidade do conflito aumenta a medida que um trabalho é mantido isolado da *codeline*
- Solução
  - Integrar as mudanças assim que estejam prontas.

P5. Fazer junção antecipada e frequente

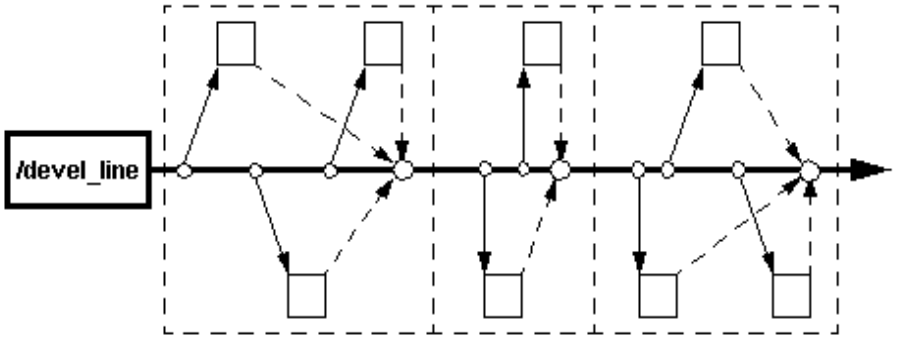


P5. Fazer junção antecipada e frequente

- Variantes
  - P5.1 Propagar antecipada e frequentemente



  - P5.2 Fazer multi-junções antecipada e frequentemente



P6. MYOC (*Merge Your Own Code*)

- Contexto
  - Após trabalhar em um *codeline*, realizar a junção para contemplar o *Merge Early and Often*
- Problema
  - Quem realiza a junção?
- Forças
  - No momento de junção o conteúdo da *codeline* pode ser diferente do conteúdo quando a modificação começou
  - O *change-owner*, *code-owner* e *codeline-owner* podem ser pessoas diferentes

P6. MYOC (*Merge Your Own Code*)

- *Code-owners* e *change-owners* que não realizam a junção do próprio código perdem noção do impacto dos seus esforços

- Solução
  - O *change-owner* ou o *code-owner* realizam a junção
    - Para tal é necessário a cooperação entre o *change-owner*, *code-owner* e o *codeline-owner*
  - Caso o *codeline* seja de alto risco, talvez possa ser necessário um outro tipo de integração

P6. MYOC (*Merge Your Own Code*)

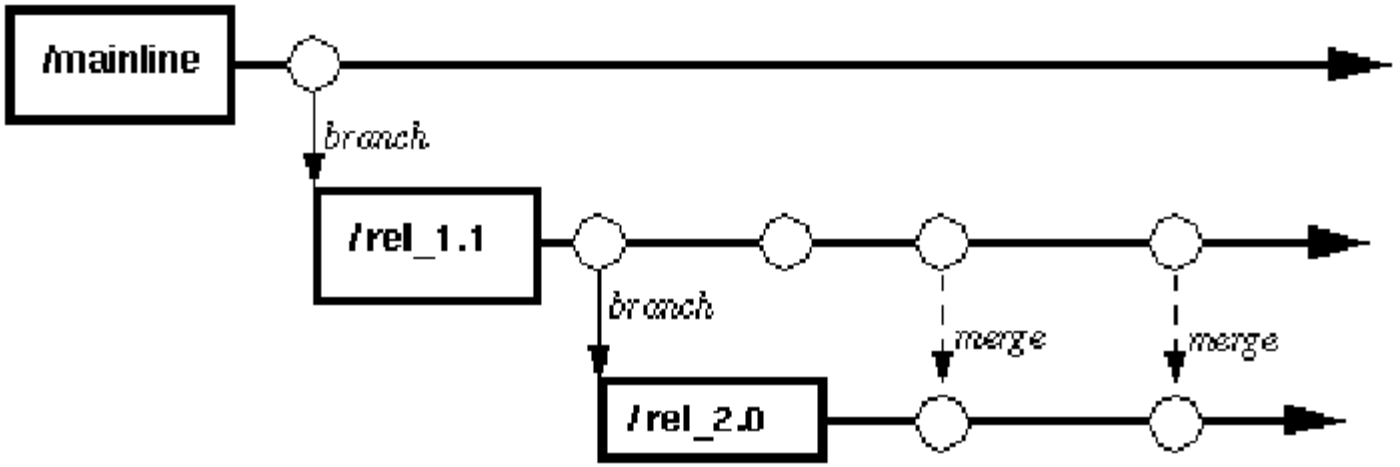
- Variantes
  - PYOC (*Propagate Your Own Code*) : Agora em vez de realizar uma junção, deve-se propagar para outra *codeline*.

P7. *Early Branching*

- Contexto
  - Futuras tarefas e subprojetos serão melhor realizados com um desenvolvimento paralelo.
- Problema
  - Devem ser criados novos ramos para estas tarefas e subprojetos
- Forças
  - Para cada ramo criado existe o risco de necessidade de junções trabalhosas
  - Isolamento de tarefas que podem ser conflituosas entre si

P7. *Early Branching*

- Solução
  - Criar um nova ramo ou *codeline* assim que uma tarefa paralela começar

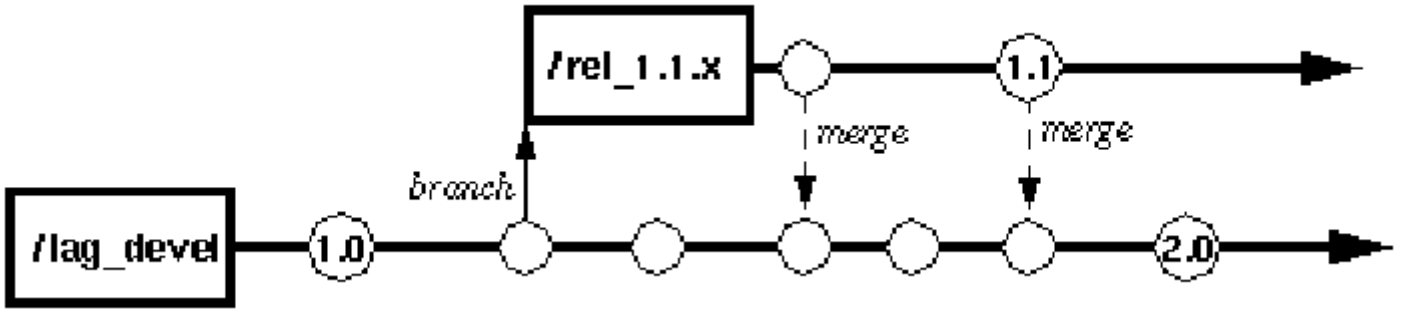


P8. *Deferred Branching*

- Contexto
  - Futuras tarefas e subprojetos serão melhor realizados com um desenvolvimento paralelo.
- Problema
  - Devem ser criados novos ramos para estas tarefas e subprojetos
- Forças
  - Para cada ramo criado existe o risco de necessidade de junções trabalhosas
  - Isolamento de tarefas que podem ser conflituosas entre si

P8. *Deferred Branching*

- Solução
  - Criar um novo ramo ou *codeline* assim que houver uma mudança lógica que cause um conflito na *codeline* original



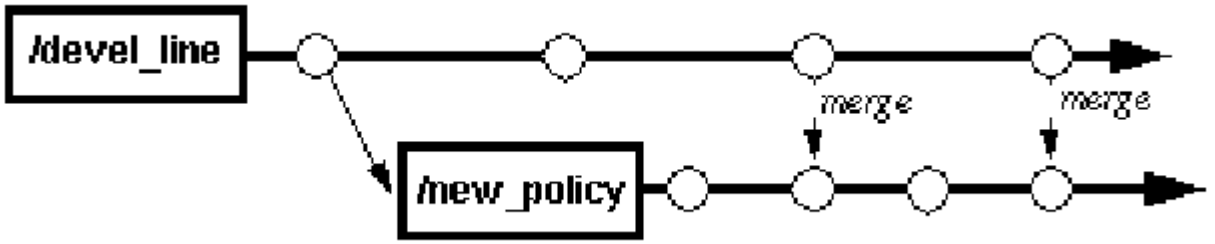
Padrões de Criação de Ramos

C1. Ramo de Política

- Contexto
  - Desenvolvedores trabalham em uma mesma *codeline* sobre uma determinada política
- Problema
  - Alguns desenvolvedores necessitam de uma nova política para a *codeline*
- Forças
  - É necessário uma nova política de *codeline* para alguns desenvolvedores enquanto que a velha política deve continuar para outros

C1. Ramo de Política

- Solução



C2. Ramo por Tarefa

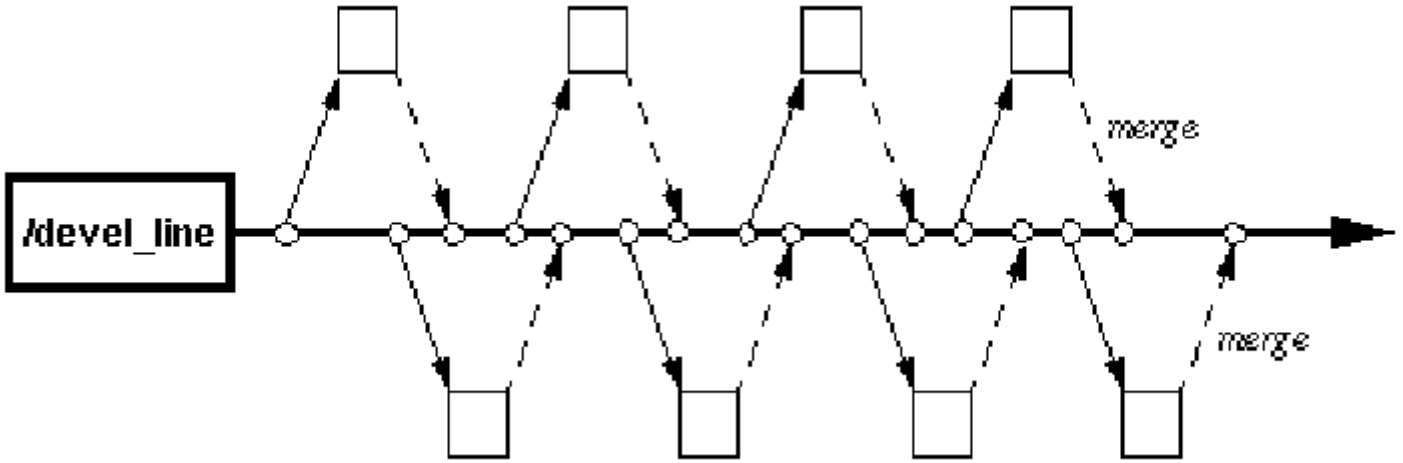
- Contexto
  - Em uma *codeline* várias tarefas podem causar problemas de sobreposição em um conjunto de arquivos
- Problema
  - Como podem várias mudanças que sobrepõem determinados arquivos serem realizadas sem comprometer a consistência e integridade do projeto?

C2. Ramo por Tarefa

- Forças
  - Desenvolvimento paralelo sem controle pode gerar em arquivos corrompidos, perdas, desperdício de esforço e retrabalho
  - Mudanças concorrentes devem ser realizadas com cuidado.
  - *Lockings* excessivos podem causar demoras no desenvolvimento e no pior dos casos *deadlock*.

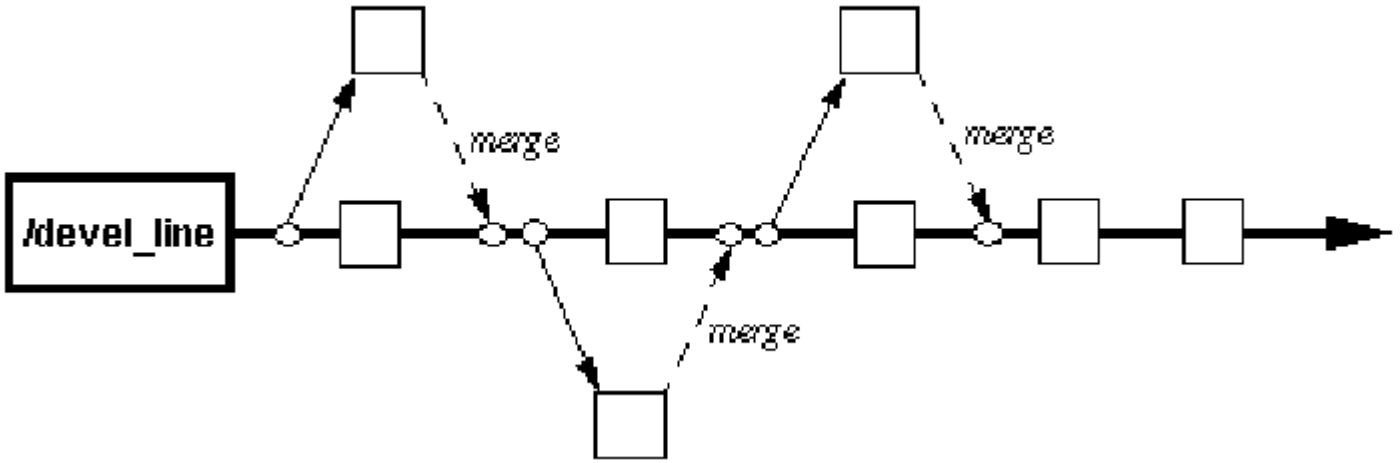
C2. Ramo por Tarefa

- Solução



C2. Ramo por Tarefa

- Variantes
  - *Branch per Major Task*



C2. Ramo por Tarefa

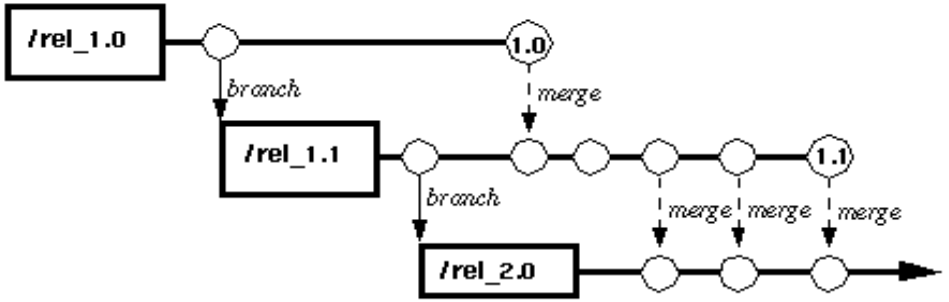
- *Branch per Change Request*: Um ramo para cada requisição de mudança

- *Personal Activity Branch*: O dono da tarefa é o dono do ramo, onde só ele desenvolve.

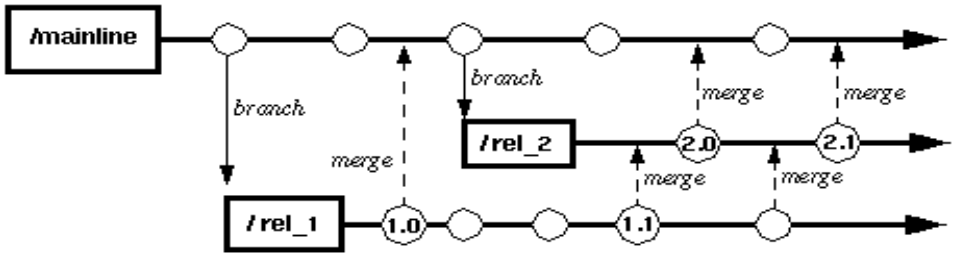
C3. Codeline por *release*

- Contexto
  - Um *software* possui várias *releases*
  - Organizar o trabalho baseado nas *releases*
- Problema
  - Como refletir as várias *releases* e os trabalhos na árvore de ramos do projeto
- Forças
  - Uso de etiquetas por *release* é insuficiente
  - Trabalhos específicos para cada *release*
- Solução
  - Usar uma *codeline* para cada *release* planejada.

C3. Codeline por release



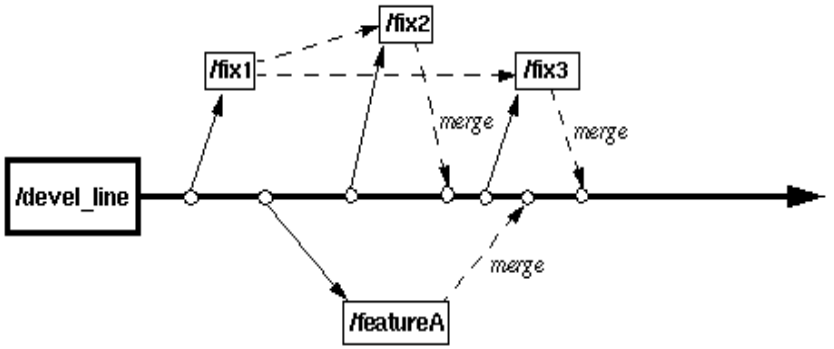
- Variantes
  - C3.1 *Codeline* por *release* principal



- Uso de uma *codeline* principal (*mainline*) é encorajado!

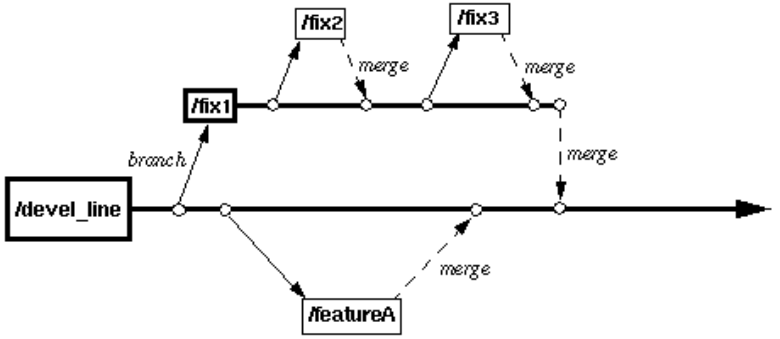
C4. Linha de subprojeto

- Contexto
  - Ramo por tarefa
  - Tarefa grande o suficiente para ser subdividida em subtarefas
  - Interdependência entre subtarefas
- Problema
  - Como organizar as subtarefas?
- Forças
  - Tarefa como uma única transação de mudança na *codeline*
  - O *checkin* de subtarefas individuais pode comprometer a integridade
  - Ordenar as subtarefas adiciona complexidade



C4. Linha de subprojeto

- Solução
  - Adicionar um nível de integração
    - Criar um ramo para a tarefa
    - Subramos para as subtarefas



- Variantes
  - C4.1 *Codeline* pessoal (ramo pessoal)
  - C4.2 Linha experimental (de protótipo)
  - C4.3 Linhas multi-projetos

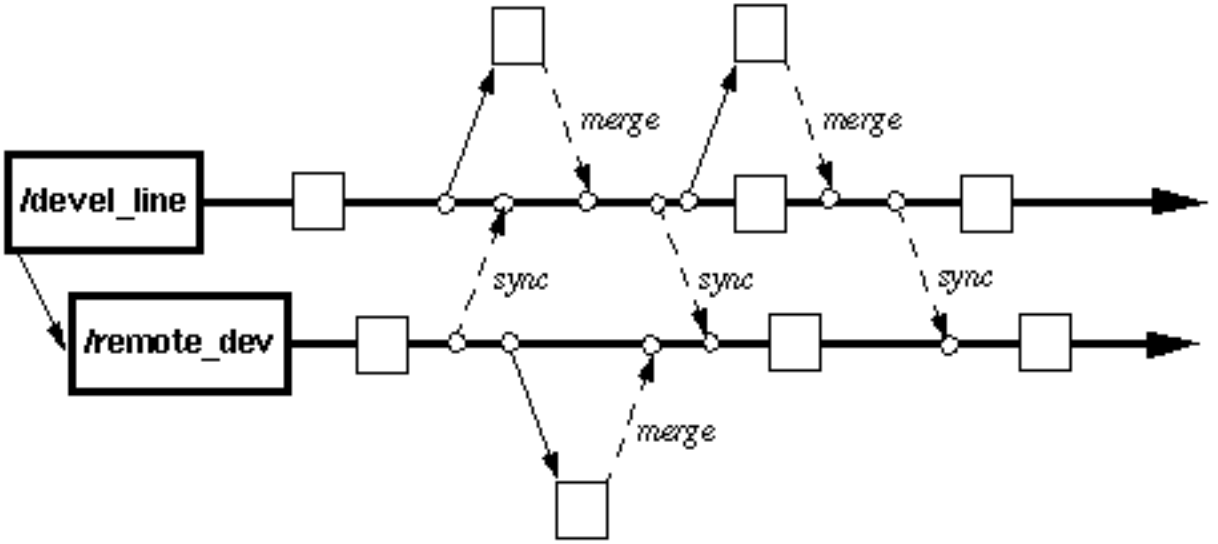
C5. *Codeline* virtual

- Contexto
  - Deseja-se utilizar ramos e *codelines* mas a ferramenta não suporta ramos com nomes;
  - Deseja-se otimizar as junções necessárias pela ramificação
- Problema
  - Como “emular” *codelines*? Como evitar junções de cópia (*copy-merge*)?
- Forças
  - Ramos melhoram a organização hierárquica das mudanças
  - Dificuldade de definir e selecionar ramos sem nomes significativos
  - Junções de cópia (espaço e retrabalho)
- Solução
  - Etiqueta flutuante para representar o ramo
    - Ramificação sob demanda (*JIT Branching*)
    - Não faz junções de cópia

C6. Linha de desenvolvimento remoto

- Contexto
  - Times locais e remotos atuando na mesma *codeline*
  - Verificação extra para integrar mudanças dos times remotos
- Problema
  - Como organizar o desenvolvimento?
    - Minimizar o impacto local (principal) sem causar problemas para times remotos
- Forças
  - Obrigações contratuais
  - Cuidados com espelhamento e criação de árvores conflitantes
  - Evitar bloqueios enquanto a verificação é feita
- Solução
  - Cada time remoto trabalha em um *codeline* remoto específico
  - Sincronização periódica

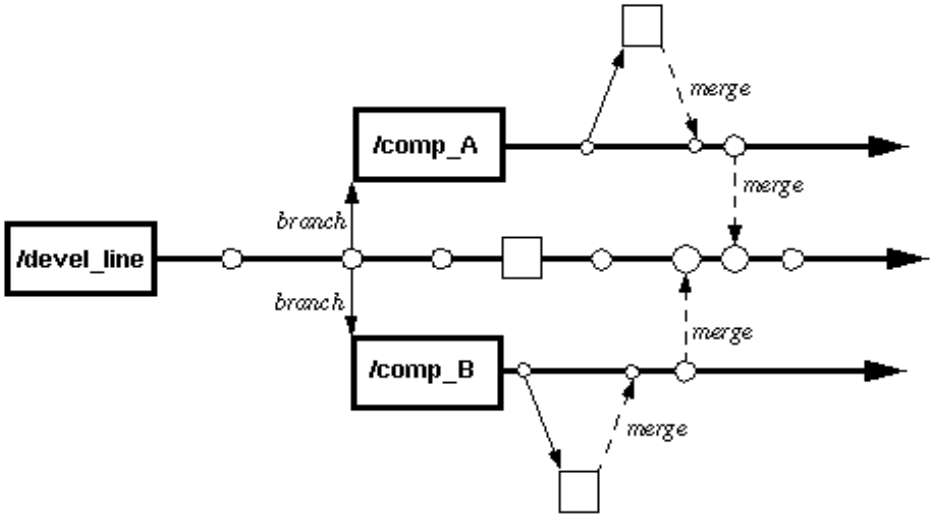
C6. Linha de desenvolvimento remoto



C7. Linha de componente

- Contexto
  - Propriedade (*ownership*) do código de componentes (módulos, arquivos etc.)
  - Múltiplas correções simultâneas e novas funcionalidades nestes componentes
  - Período de tempo pequeno
- Problema
  - Como manter os benefícios da propriedade do código mesmo com outras pessoas que não o proprietário realizando mudanças
- Forças
  - Se todos são responsáveis, então ninguém é responsável
  - Não permitir que outros mudem pode atrasar *releases*
  - Atrasos causados pela necessidade de autorizações do proprietário
- Solução
  - Criar *codeline* para este componente, colocando o proprietário do código como proprietário da *codeline*
  - Versões estáveis para o resto do projeto

C7. Linha de componente



- Variantes
  - C7.1 Linhas multi-produtos
    - Igual a C4.3 Linhas multi-projetos

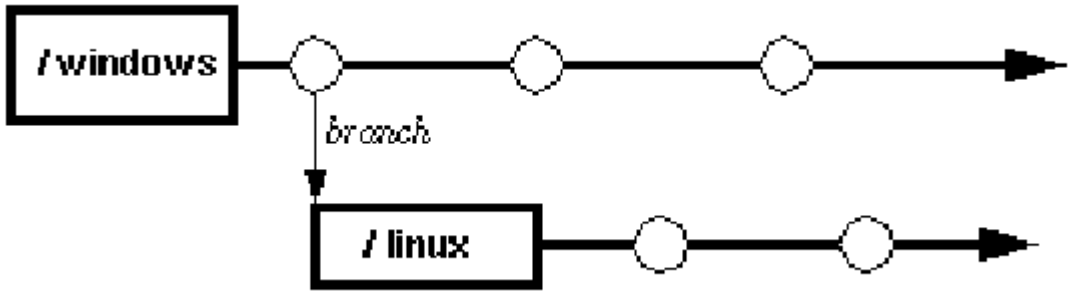
C8. Linha de Plataforma

- Contexto
  - Desenvolvimento para múltiplas plataformas
- Problema
  - Como controlar as particularidades de uma determinada plataforma em um projeto multiplataforma?
- Forças
  - Informações de construção e tempo de execução devem ser facilmente discerníveis para cada plataforma

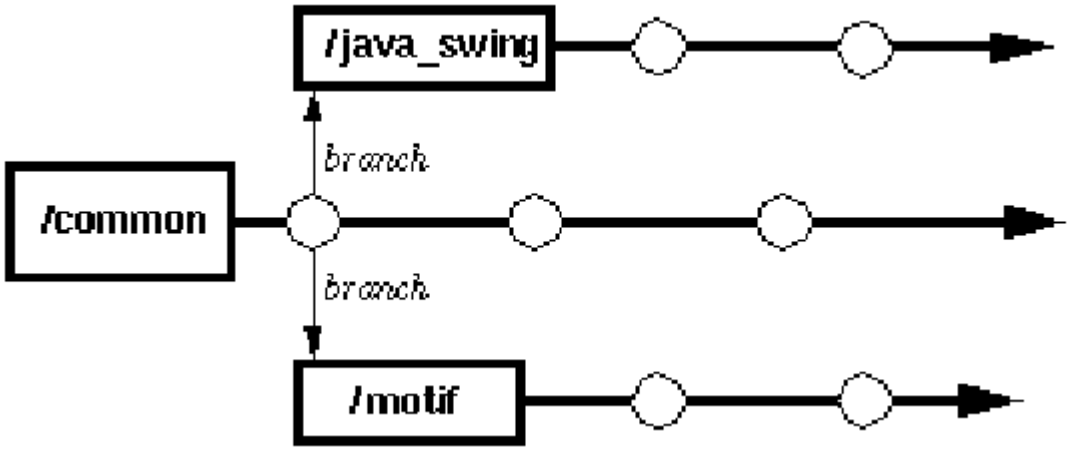
C8. Linha de Plataforma

- Mudanças para plataformas específicas não devem afetar as outras

- Solução



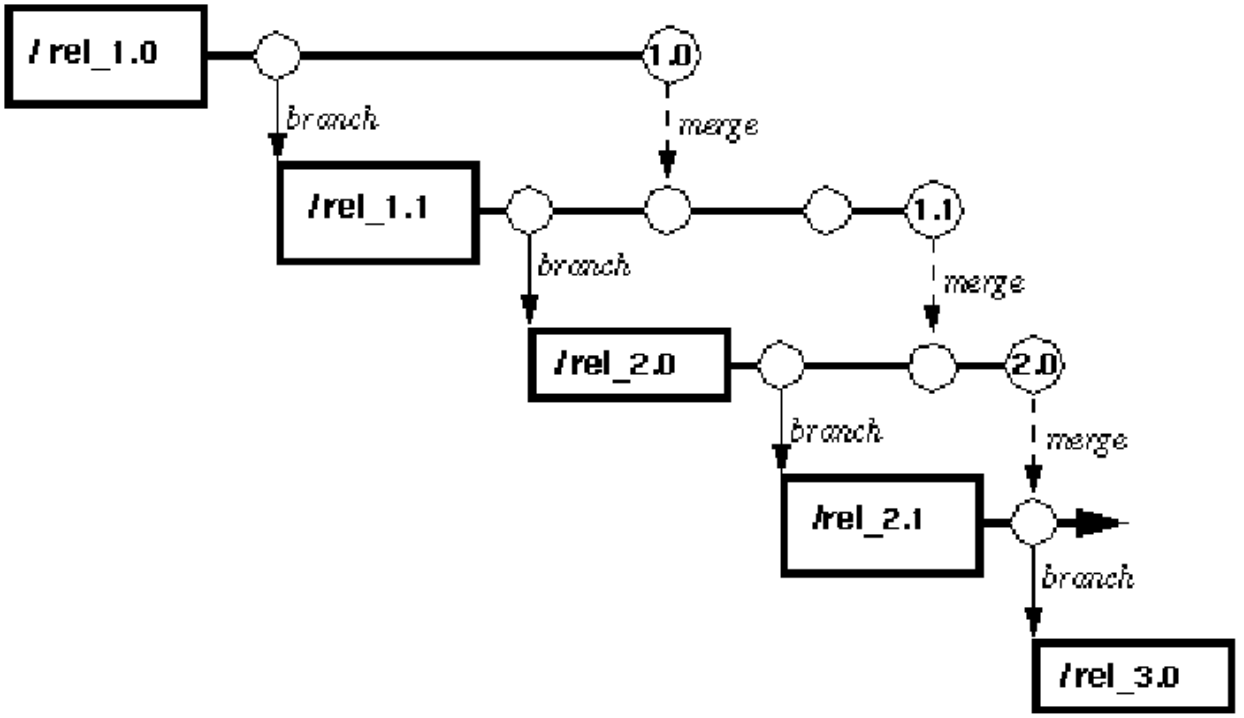
C8. Linha de Plataforma



Padrões de Estruturação de Ramos

S1. Linha Principal

- Contexto
  - Múltiplas *codelines* são criadas por variados motivos

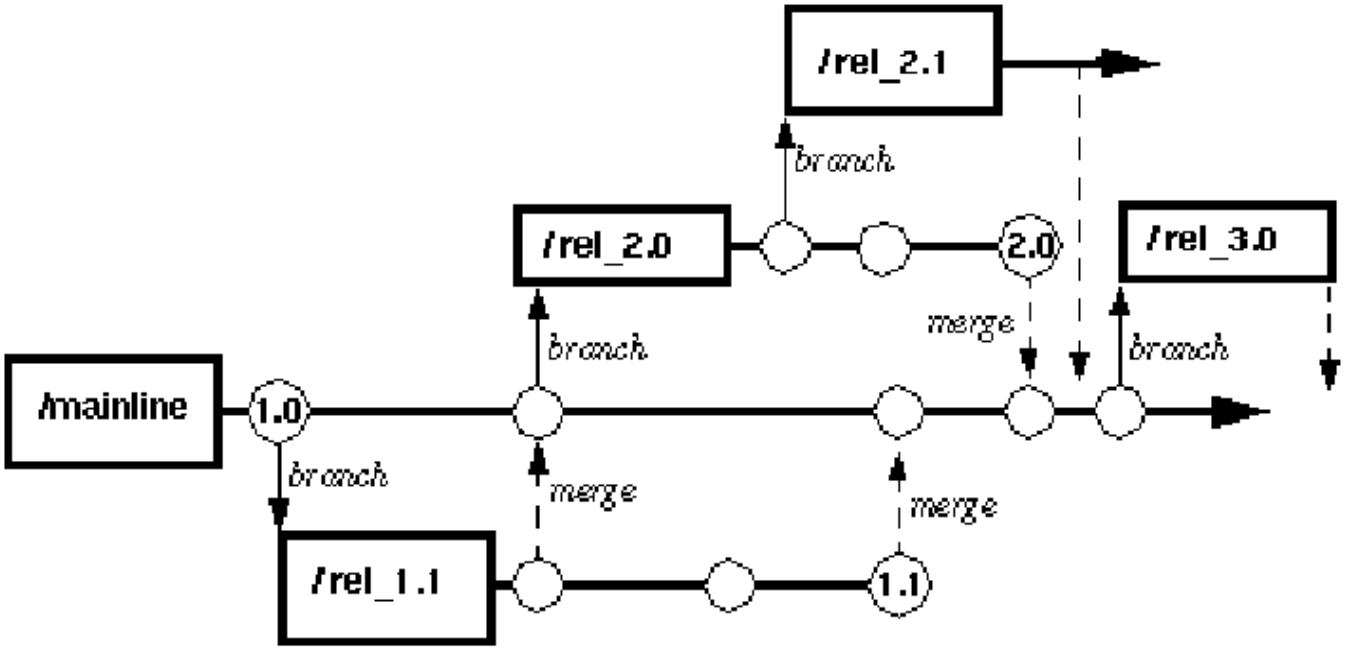


S1. Linha Principal

- Problema
  - Como manter o número de *codelines* em um conjunto gerenciável
- Forças
  - Cada *codeline* que é criada necessitará de junções, o que faz com que aumente os esforços de sincronização

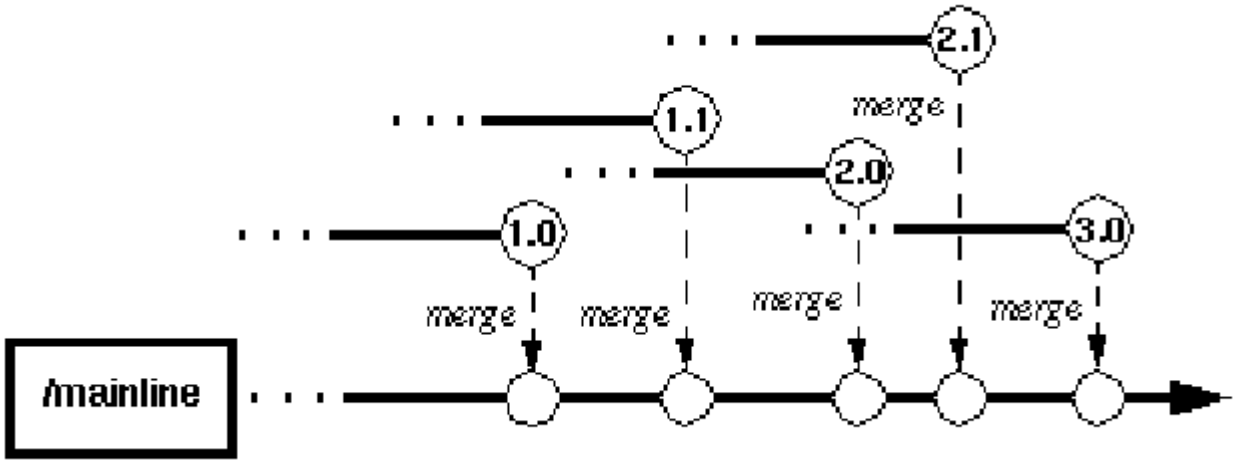
S1. Linha Principal

- Solução
  - Criar uma *codeline* principal e fazer com que as novas *codelines* girem em torno desta *codeline*



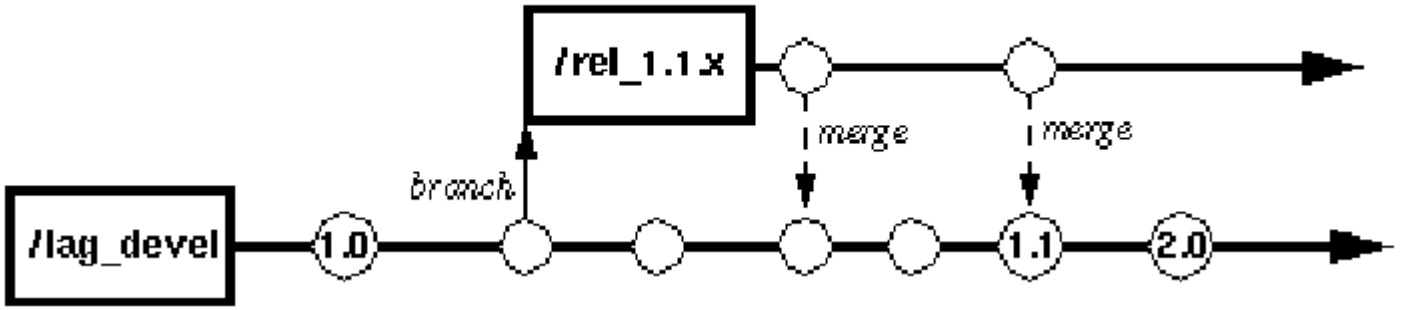
S1. Linha Principal

- Variantes
  - *Stable Receiving-Line*



S1. Linha Principal

- *LAG Development-Line*

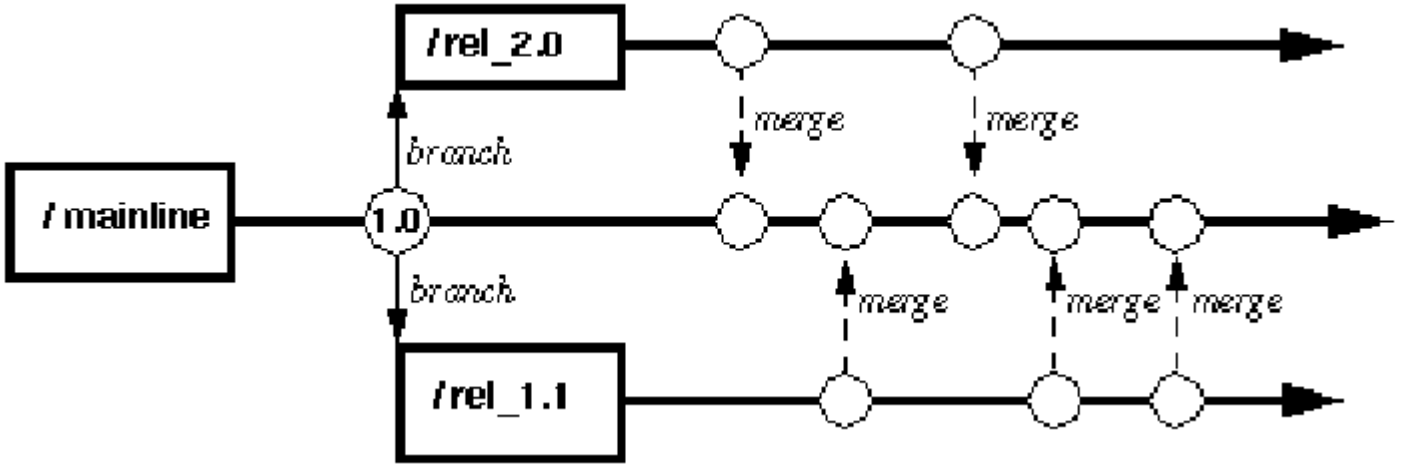


S2. Manutenção Paralela/Linhas de Desenvolvimento

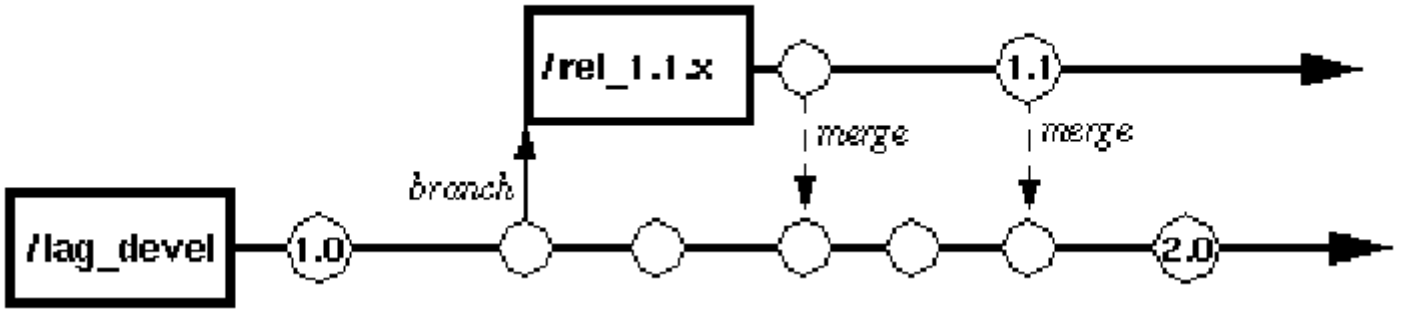
- Contexto
  - Houve uma liberação de uma versão do projeto e é necessário continuar o desenvolvimento para a próxima versão.
- Problema
  - Como continuar o desenvolvimento da próxima versão e realizar a manutenção da versão anterior
- Forças
  - Necessidade de implementar novas funcionalidades para a próxima versão

S2. Manutenção Paralela/Linhas de Desenvolvimento

  - Necessidade de corrigir os *bugs* da versão anterior
- Solução

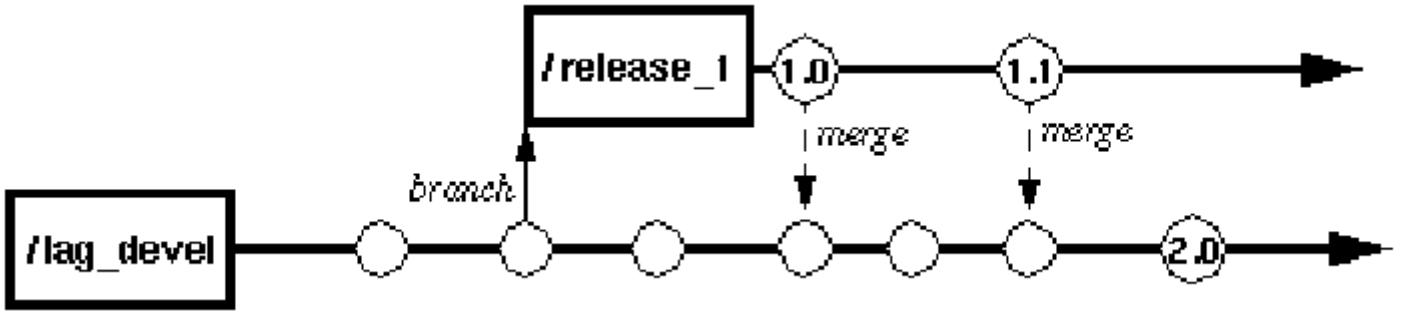


S2. Manutenção Paralela/Linhas de Desenvolvimento



S2. Manutenção Paralela/Linhas de Desenvolvimento

- Variantes
  - *Parallel Releasing/Development Lines*

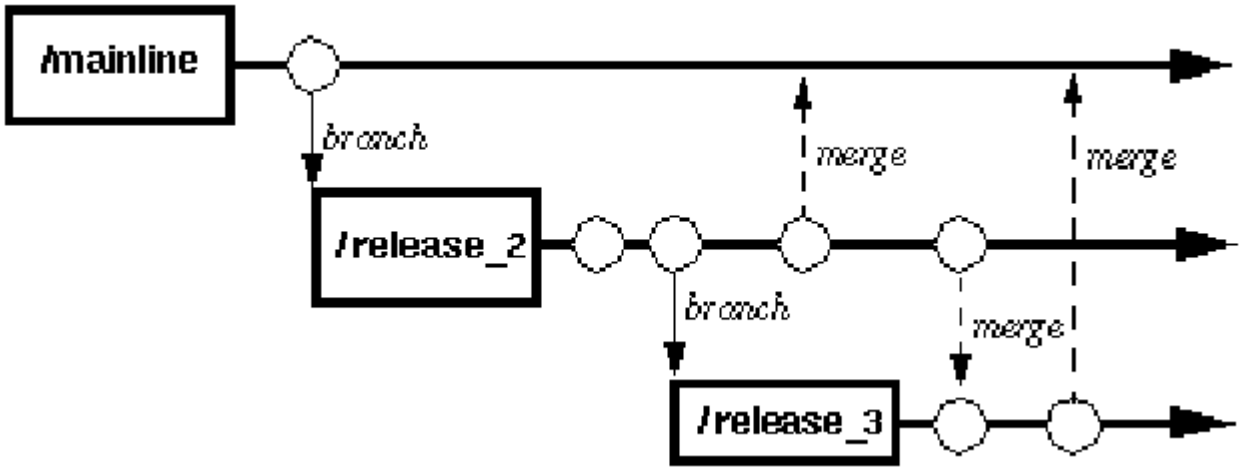


S3. Linhas de *Releases* Sobrepostos

- Contexto
  - Necessidade de se criar duas versões seguidas com um curtíssimo espaço de tempo entre elas.
- Problema
  - Como desenvolver duas versões sem que uma impacte na outra
- Forças
  - Pouco tempo para se desperdiçar

S3. Linhas de Releases Sobrepostos

- Solução



S4. Linha de acoplamento

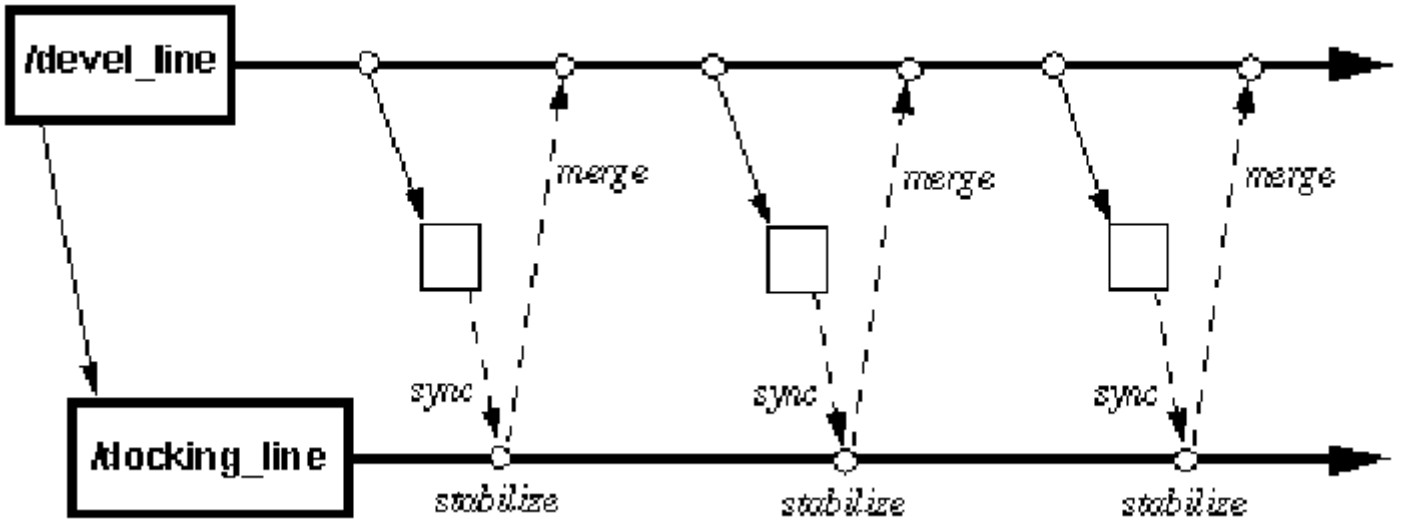
- Contexto
  - Após trabalhar em uma *codeline*, realizar a junção. A *codeline* possui um alto risco ou complexidade de desenvolvimento.
- Problema
  - Quem realiza a junção
- Forças
  - No momento de junção, o conteúdo da *codeline* pode ser diferente do conteúdo quando a modificação começou

S4. Linha de acoplamento

- O *change-owner*, *code-owner* e *codeline-owner* podem ser pessoas diferentes
- *Code-owners* e *change-owners* que não realizam a junção do próprio código perdem noção do impacto dos seus esforços
- Porém, a alta criticidade da *codeline* requer que o *codeline-owner* realize a junção

S4. Linha de acoplamento

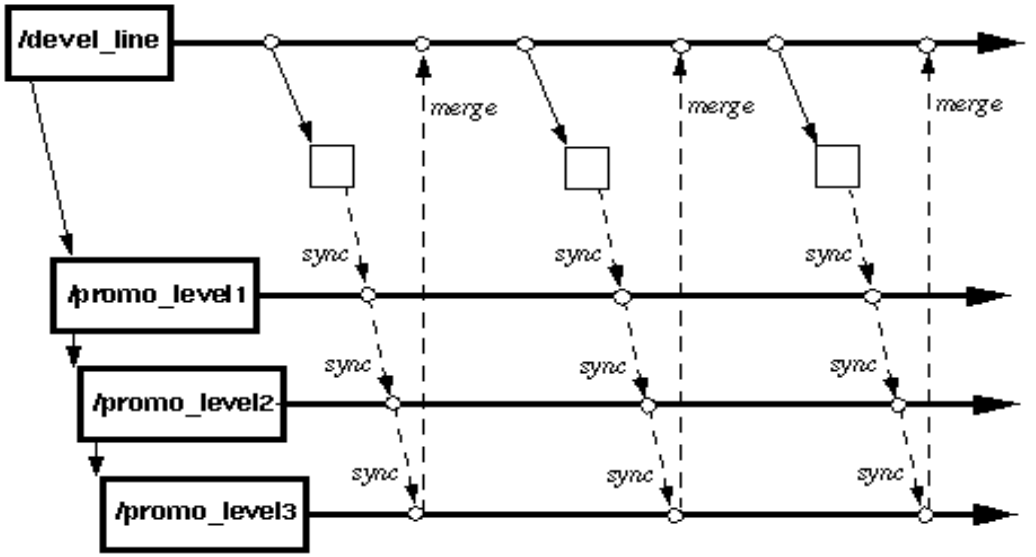
- Solução



S5. Linhas de integração por estágios

- Contexto
  - Mudanças possuem estágios (teste de unidade, de integração etc).
  - Múltiplos níveis de propriedade
- Problema
  - Como usar as funcionalidades da ferramenta para rastrear e controlar mudanças através dos vários estágios?
- Forças
  - Correspondência com eventos: revisão, testes de unidade etc., ou garantia da qualidade
  - Alguns níveis irão requerer integração
- Solução
  - Algumas ferramentas já preveem estágios (etiquetas, ramos etc)
  - Estágios → ramos de integração
  - Transferência de responsabilidades (propriedade)

S5. Linhas de integração por estágios



- Etiquetas podem ser utilizadas para representar os estágios
  - Representar estágios de qualidade e maturidade
  - Facilitam um *back-out*
  - Diminuem o *overhead* de junção

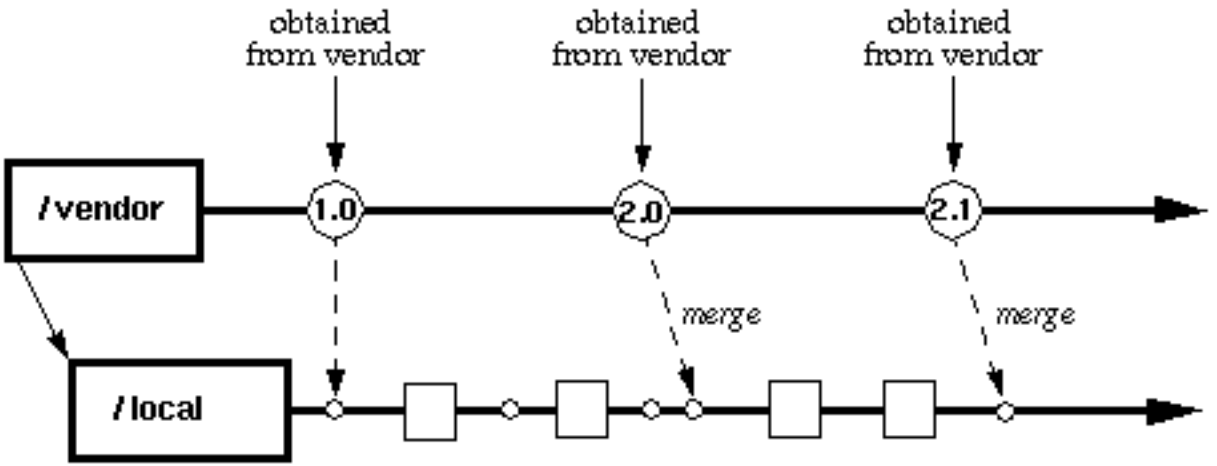
S6. Filas de propagação de mudanças

- Contexto
  - Linhas de desenvolvimento/manutenção paralelas e/ou várias linhas de *release* ativas
  - Necessidade de propagar mudanças de uma *codeline* para outra
- Problema
  - Como propagar de forma consistente e em uma ordem apropriada?
- Forças
  - Integrar mudanças o quanto antes, sem impactar outros desenvolvedores
  - Ordem é importante e o número de mudanças torna difícil manter o mapeamento sem auxílio de ferramentas
  - Nem sempre o desenvolvedor que modificou pode propagar a mudança
- Solução
  - Fila de recebimento de propagação de mudanças para a *codeline*
    - Auxílio da ferramenta de controle de mudanças e da ferramenta de controle de versões
    - *Scripts*
- Variantes
  - S6.1 Auto propagação e enfileiramento

S7. *Codeline* de terceiros

- Contexto
  - Pacotes de código são recebidos de um terceiro
  - Entregar o pacote com customizações aos clientes
- Problema
  - Estratégia para lidar com atualizações do terceiro e mudanças para customização
- Forças
  - Minimizar o esforço manual requerido
  - Reproduzir *releases* anteriores
  - Isolar as customizações
- Solução
  - Usar ramificação para manter separado o que é recebido do que é entregue ao cliente
  -

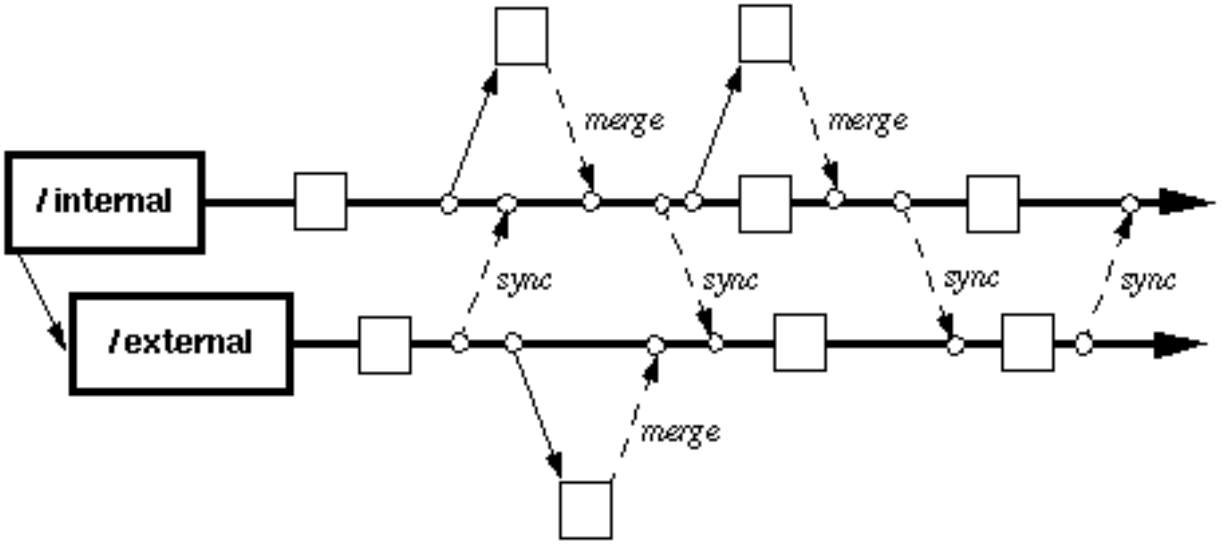
S7. *Codeline* de terceiros



S8. Linhas interna e externa

- Contexto
  - Times remotos com acesso ao repositório para efetuar mudanças em uma *codeline*
- Problema
  - Fornecer acesso sem comprometer a consistência e a integridade da *codeline*
- Forças
  - Acessível por um número considerável de desenvolvedores (alguns não confiáveis)
  - Travas de *checkout* podem ser intratáveis
  - Controles devem ser implementados
- Solução
  - Usar uma *codeline* interna para o time local (confiável) e outra separada para receber os times externos (com restrições de acesso)
  - Sincronizações são feitas pelo proprietário da *codeline* interna
  - A propriedade da *codeline* externa deve ser do alguém do time local, para manter a consistência com as políticas das duas *codelines*.
  - A *codeline* externa atua como *firewall*

S8. Linhas interna e externa



Como utilizar

- Selecionar o estilo de ramificação
  - Antecipado ou adiado
    - Antecipado: mais formal, maior isolamento e controle
    - Adiado: maior produtividade, menos isolamento, maior risco
- Selecionar o estilo de junção
  - Relaxado ou restrito
    - Relaxado: mais ativo, mais arriscado.
    - Restrito: mais seguro, gera mais *codelines*
  - Pode haver mesclas no projeto

Como utilizar

- Começar simples
  - Evitar trabalhos desnecessários
- Mudar as escolhas se necessário
  - Variantes
  - Mais controle (relaxado para restrito)
- Considerar as necessidades do projeto

Conselhos Gerais

- Usar nomes significativos para os ramos
- Ramificar quando necessitar realizar um "*freezing*"
- Integre antecipadamente e frequentemente
- Ramificar quando houver forças competindo entre si
- Adicionar outro nível de integração
- Manter a simplicidade

Conselhos Gerais

- Preservar a integridade e consistência
- Isolar as mudanças
- Isolar o trabalho e não as pessoas

Contexto resultante

- Similaridades com a programação paralela
- Isolamento e mitigação de riscos
- Complexidade de gerenciamento com hierarquia
- Custos de integração
- Integridade e reprodutibilidade
- Comunicação, coordenação e produtividade
- Paralelização com construções concorrentes

Contexto resultante

- Topologia da ramificação representa a estrutura do *workflow* de atividades do projeto.
- Alinhamento arquitetural

Obrigado!