Padrões em Gerência de Configuração

Reuso de soluções

Créditos

- Compilação de padrões contidos no livro "*Software Configuration Management Patterns: Effective Teamwork, Practical Integration*" de Steve Berczuk

http://www.scmpatterns.com/book/

O que são padrões?

- Padrões são formas de representação de conhecimento
  - organizados de forma estruturada
- Objetivam a rápida assimilação e aplicação em um novo contexto
- Construídos a partir da destilação de anos de experiência

Estrutura de um padrão

- Nome
- Contexto
- Problema a ser solucionado
- Solução
- Diagrama representando a solução (opcional)

Padrões em GC

- Catálogo com 15 padrões
- Classificação dos padrões:

Codeline	Workspace

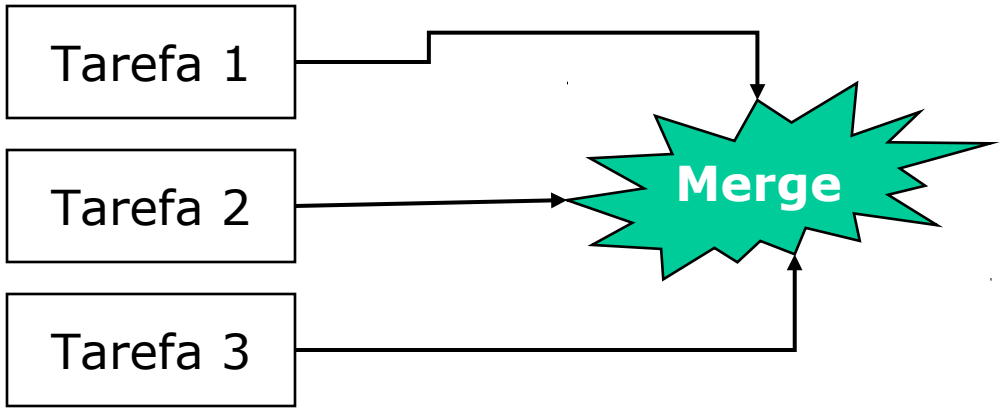
Mainline
"Simplifique seu modelo"

Classificação: Codeline

- Você quer simplificar sua estrutura de codelines
- **Como manter várias codelines (e minimizar merging)?**

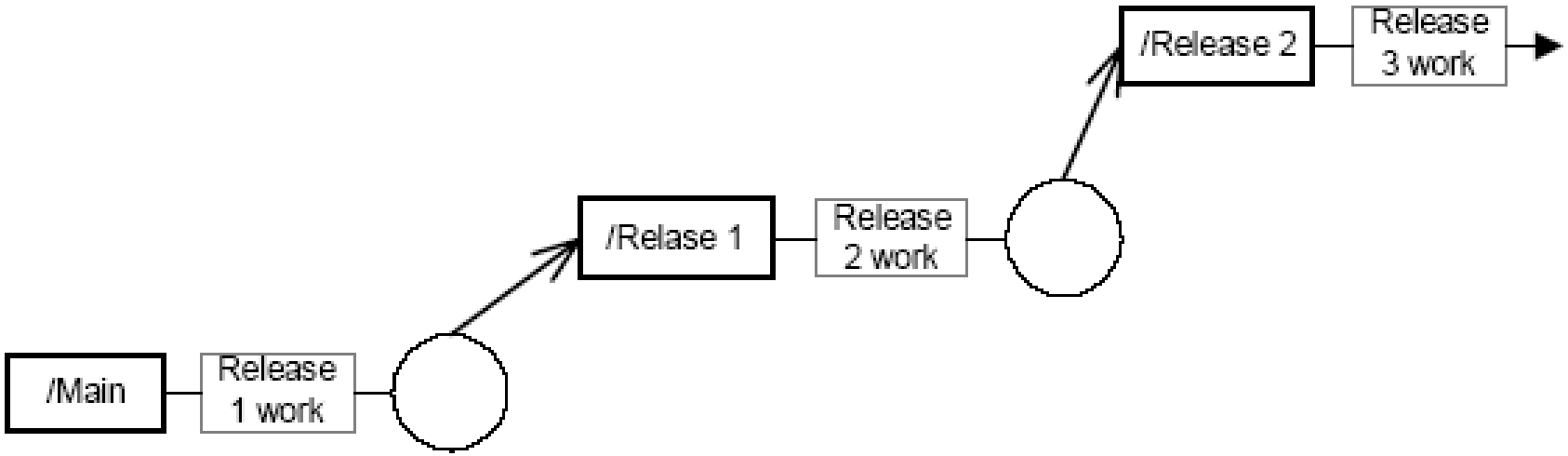
Mainline (Contexto)

- A utilização de *Branches* 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)

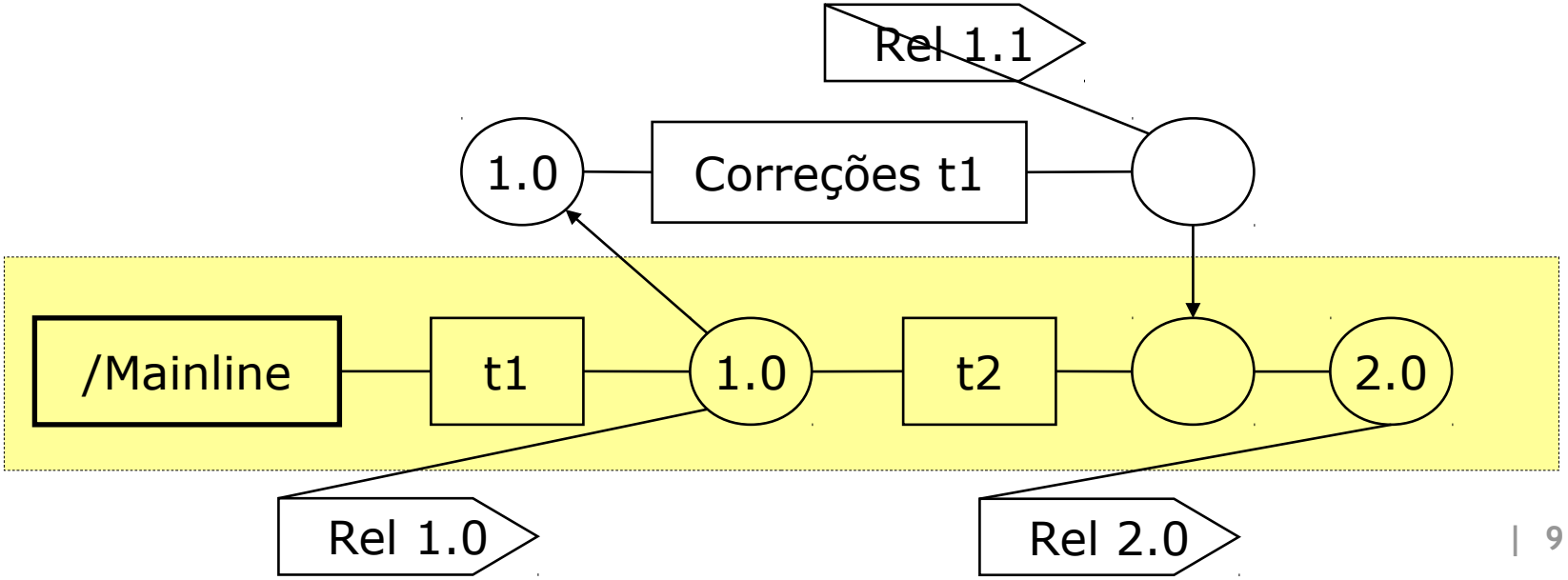
- O uso desenfreado de *branches* pode gerar estruturas complexas, difíceis de manter e integrar



- Algumas empresas aboliram o uso de *branches* devido a más experiências

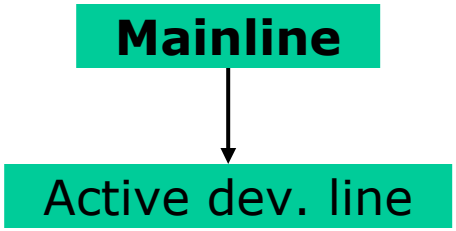
Mainline (Solução)

- **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*



Mainline (Criação/Manutenção)

1. Crie uma *codeline* a partir da base de artefatos
2. *Check-in* todas mudanças nesta *codeline*
3. Certifique-se do bom funcionamento da *Mainline* antes de fazer *checkins*
   - Teste com frequência!
4. Planeje cuidadosamente a necessidade de *e* a vida (criação e morte) de cada *branch*
5. [Não Resolvido] Como motivar o uso da *mainline* por muitas pessoas?



O padrão *Active dev. Line* é aplicado no contexto do *mainline*

Active Development Line
"*Defina suas metas*"

- *Classificação: Codeline*

- Você está desenvolvendo em uma mainline
- **Como manter uma codeline em rápida evolução estável o suficiente para ser útil?**

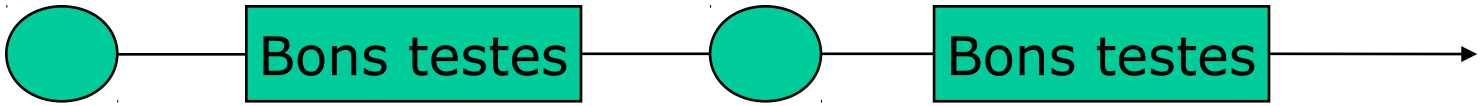
Active Development Line (Contexto)

- 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 afetam a todos

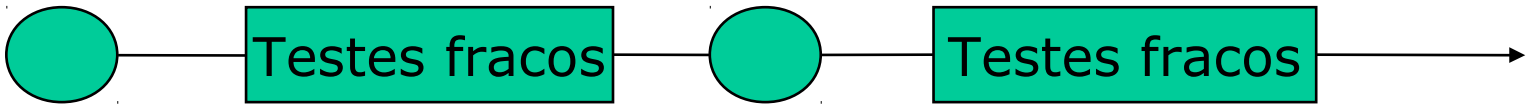
- Procedimentos rigorosos e rígidos
  - Demoram muito tempo
    - Mais ainda se houver muitos desenvolvedores
  - Geram menos check-ins (natureza humana)
  - Não aplicam-se em todos os casos

Active Development Line (Contexto)

- Problemas de produtividade
  - Linha de desenvolvimento pouco ativa e muito estável

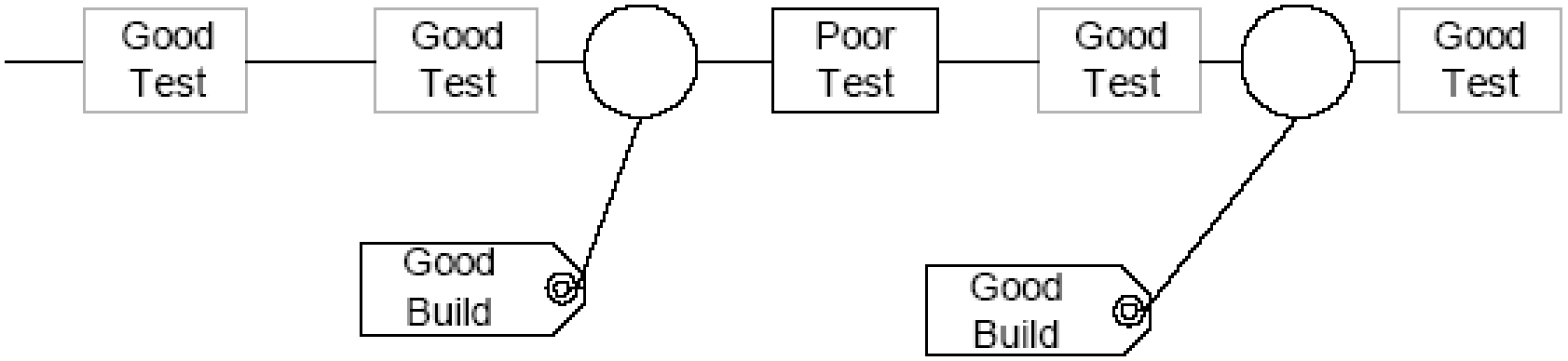


  - Linha de desenvolvimento muito ativa e pouco estável



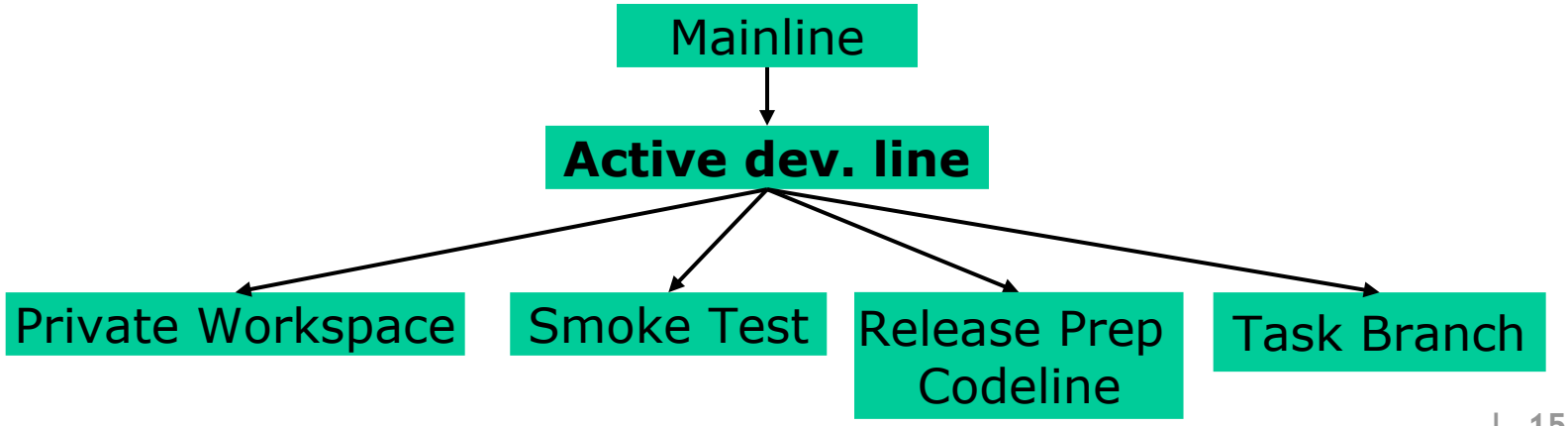
Active Development Line (Solução)

- **Use uma linha de desenvolvimento ativa**
  - Com políticas para a estabilidade necessária da codeline para suas necessidades
  - Considerando o ritmo de desenvolvimento do projeto



Active Development Line (Problemas não resolvidos)

- Alguns padrões complementares (necessários!)
  - Private Workspace : Ambiente isolado de desenvolvimento
  - Smoke Test : Mantém a estabilidade da codeline
  - Task Branch: Lida com tarefas de grande risco:
  - Release Prep Codeline: Evita congelamento da codeline



Private Workspace
"Isole o seu trabalho"

- *Classificação:* Workspace

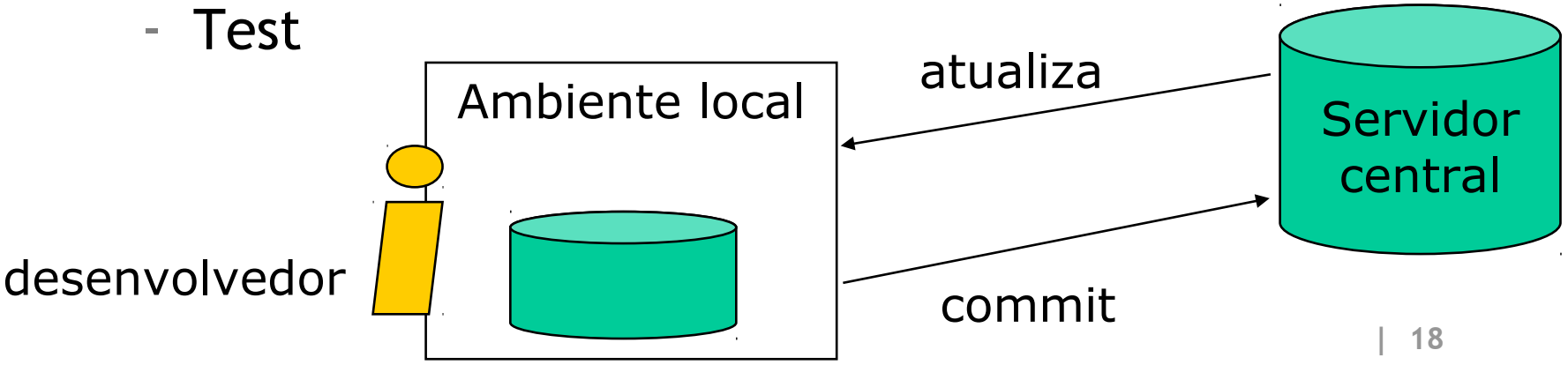
- Você quer dar suporte a uma Active Development Line
- **Como evoluir sem se distrair com as freqüentes mudanças do ambiente?**

Private Workspace (Contexto)

- Integração freqüente evita o trabalho com artefatos desatualizados
  - Porém, pessoas não conseguem raciocinar em um ambiente em constante mudança
- Algumas vezes, tarefas não-relacionadas precisam ser realizadas
  - Porém, isolamento em excesso é proibitivo

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

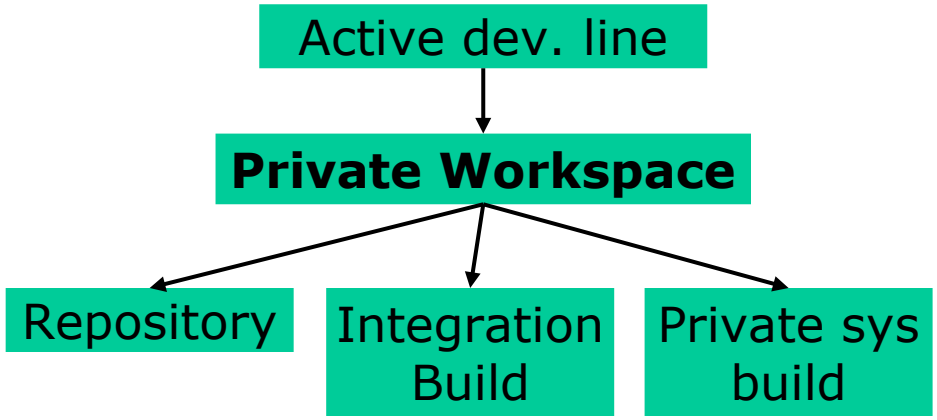


Private Workspace

- Pode conter:
  - Código-fonte a ser editado
  - Componentes compilados localmente
  - Dados e código para testar localmente o sistema
  - *Scripts* de *build*
- Não pode conter:
  - Versões especializadas de scripts, quebrando a política do projeto;
  - Componentes que estão versionados mas que você copiou de um outro local desconhecido;
  - Diferentes versões de ferramentas certificadas pela organização

Private Workspace (Problemas não resolvidos)

- Alguns padrões complementares
  - Repository: Povoar o ambiente de trabalho
  - Private system build: Construir e testar o ambiente
  - Integration Build: Integração com as mudanças dos outros



Repository
"Um único lugar para compras"

- *Classificação:* Workspace

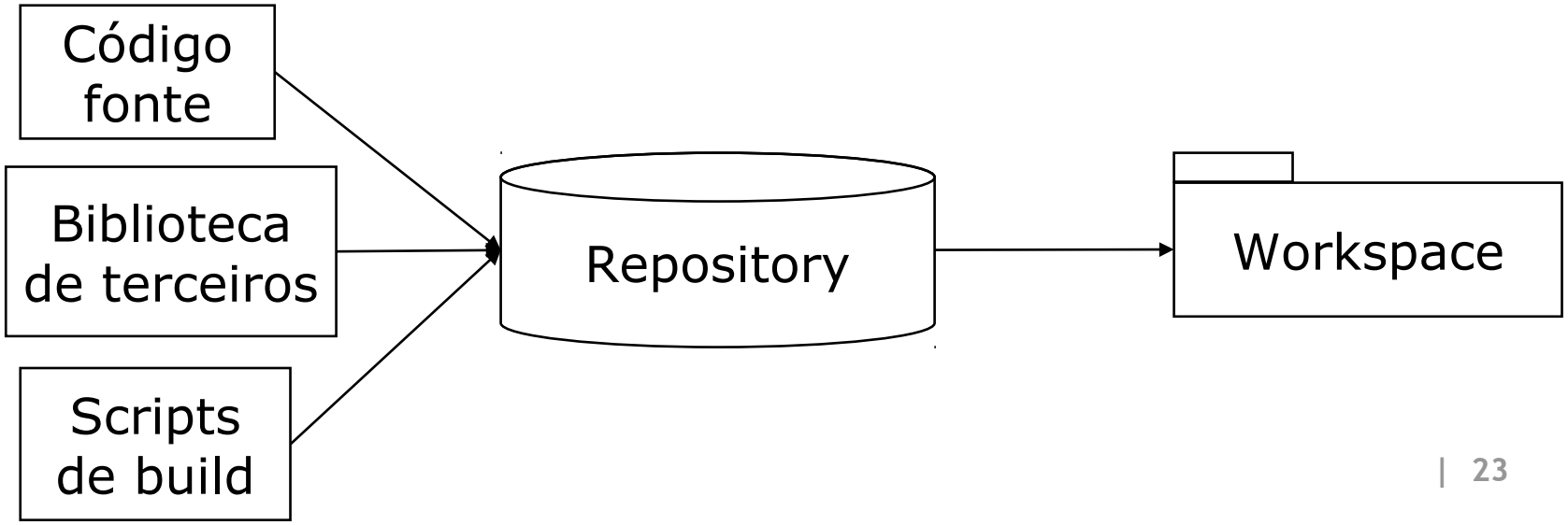
- Private workspaces e Integration Builds precisam de componentes
- **Como obter a versão correta de um determinado componente em um ambiente de trabalho?**

Repository (Contexto)

- Diferentes artefatos compõem um workspace
  - Codigo, scripts, componentes de terceiros
- Artefatos podem vir de diversas fontes
  - Outros grupos, empresas terceiras
- Integradores, Testadores e Desenvolvedores precisam, se necessário, ter acesso às mesmas versões de componentes

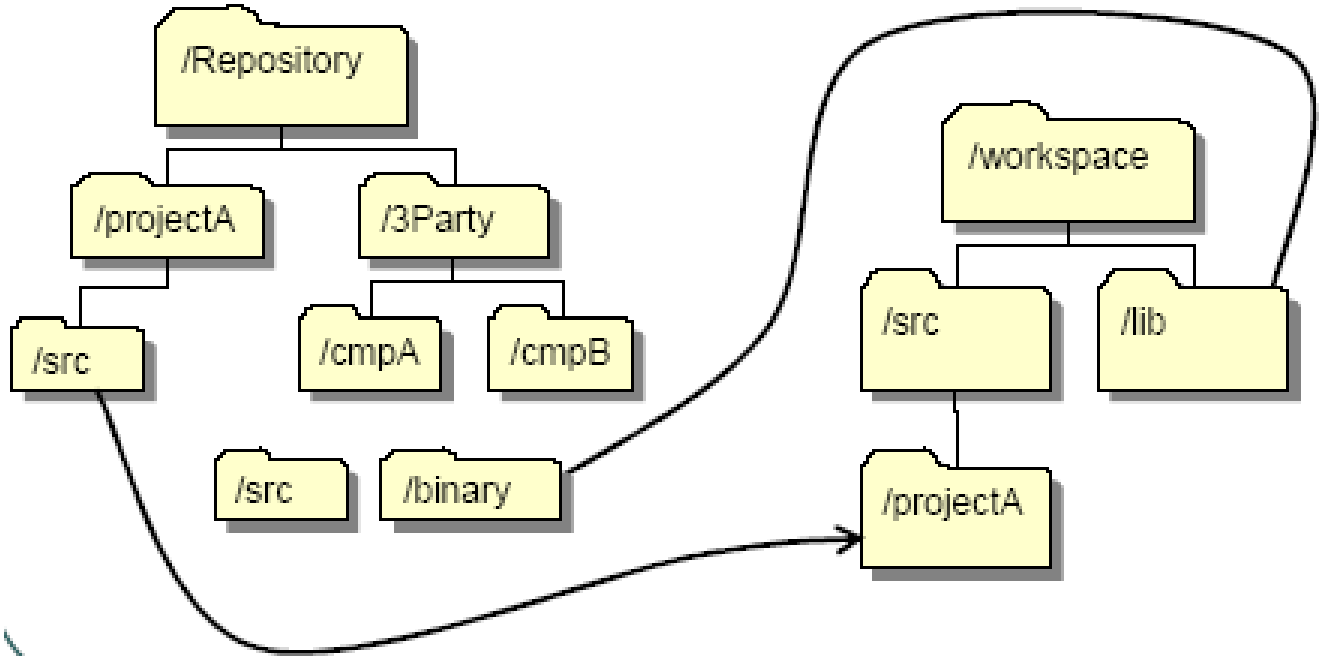
Repository (Solução)

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



Repository (Problemas restantes)

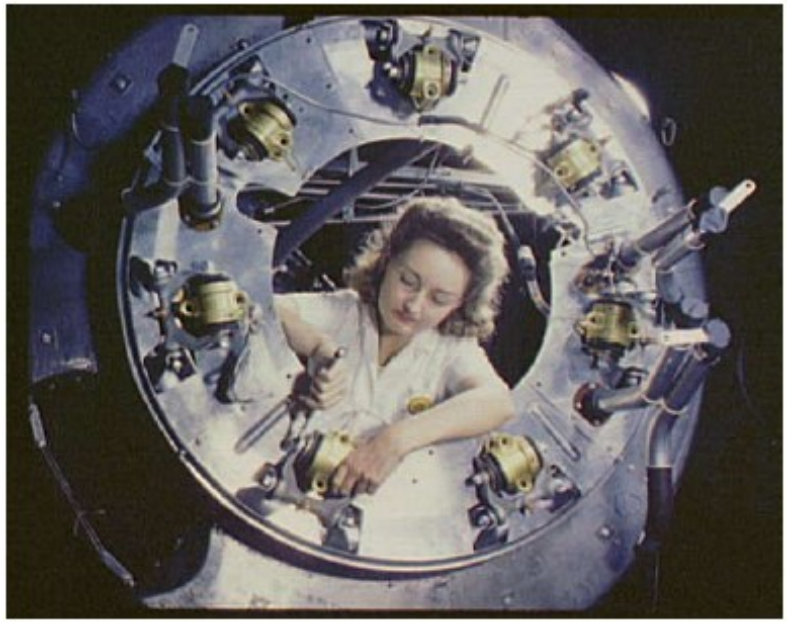
- Mapear os componentes do repositório para o ambiente de trabalho
  - Uma ferramenta para geração de *builds* pode fazer isso



Private System Build

"Pense globalmente e construa localmente"

- *Classificação:* Workspace



- Você precisa construir e testar o que está em seu *Private Workspace*
- **Como verificar se suas mudanças não quebram o sistema antes de realizar um *commit* no Repositório?**

Private System Build (Contexto)

- Adequar as necessidades de um *build* para um *workspace* de desenvolvimento
  - O *build* do sistema pode ser complicado
  - Manter consistência com outros builds
- Evitar *commits* de mudanças que quebrem o sistema
  - Atenuar isolamento de um Private Workspace

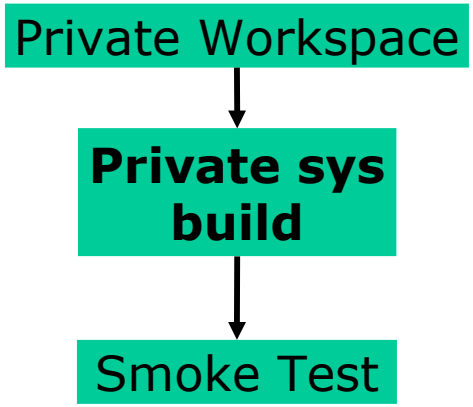
Private System Build (Solução)

- Construa o sistema usando os mesmos mecanismos do build central de integração
  - Use os mesmos componentes e ferramentas (reproduzir problemas)
  - Considere apenas detalhes relevantes ao desenvolvimento
  - Inclua todas as dependências requeridas
- Faça isto antes de cada commit
  - Inclua chamada para testes durante o build
- Atualize o cabeçalho da codeline antes de executar o build

Private System Build (Problemas não resolvidos)

- Se a construção do sistema todo é proibitiva, construa a menor quantidade de componentes que sua mudança requer.

- Alguns padrões complementares
  - Smoke Test: Testar o que você construiu



Integration Build
"Realize um build centralizado"

- *Classificação:* Workspace

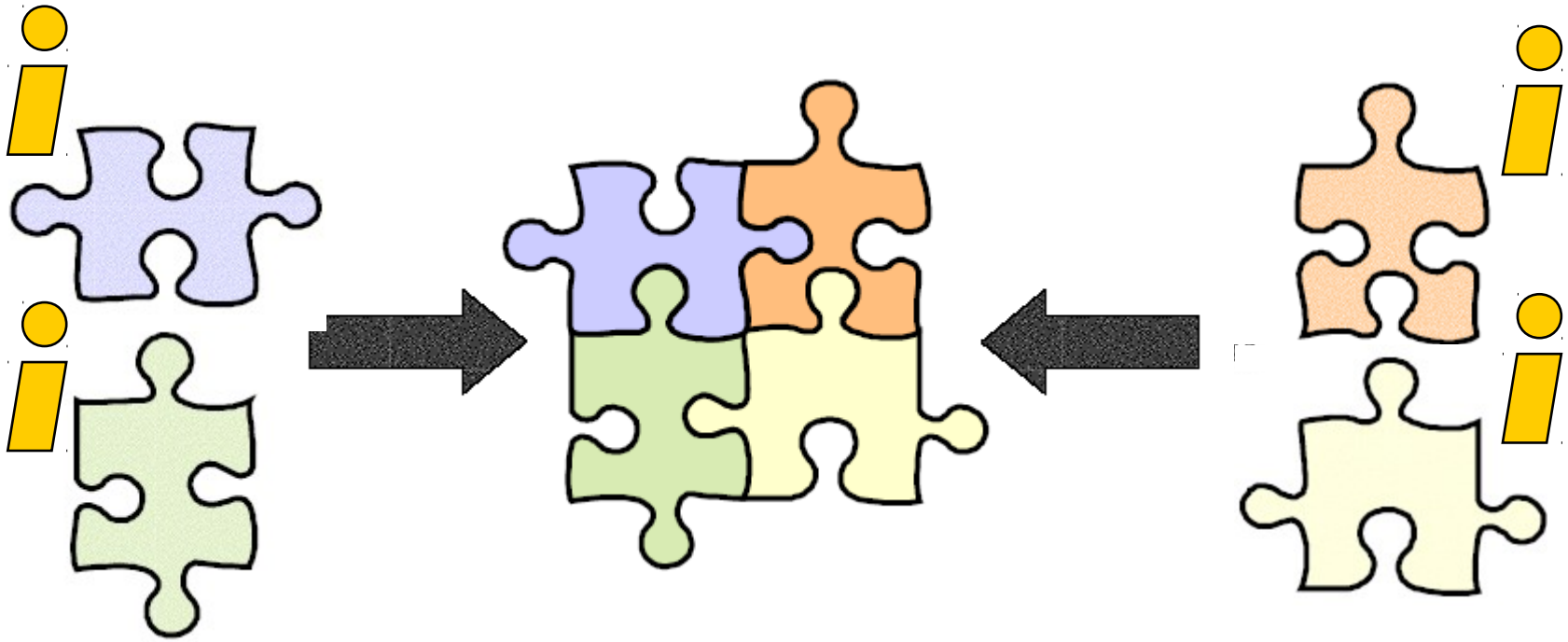
- O que é realizado em um Workspace precisa ser compartilhado
- **Como garantir que o codebase sempre é construído consistentemente?**

Integration Build (Contexto)

- Trabalhos individuais devem ser integrados
  - Comunicação não é o suficiente
- Você quer garantir que o que é atualizado no repositório funciona
- Private System Build é uma forma de testar o sistema
  - Porém, construir sempre todo o sistema pode levar muito tempo
  - Construir localmente **não garante** corretude

Integration Build (Solução)

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



Integration Build (Processo)

- Determine a freqüência 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
  - Registrado em um log e problemas devem ser notificados via email

[nightly] 23-Mar-2012 build of STABLE on i386-unknown-linux (cam-02-unx)

Entrada x

GHC Build Reports ghcbuild@microsoft.com por haskell.org 24/03/12

para cvs-ghc

inglês > português Traduzir mensagem Desativar

Build description = STABLE on i386-unknown-linux (cam-02-unx)
Build location = /playpen/simonmar/nightly/STABLE
Build config file = /home/simonmar/nightly/site/msrc/conf-STABLE-cam-02-unx

Nightly build started on cam-02-unx at Fri Mar 23 18:10:01 GMT 2012.
**** checking out new source tree ... warning: Remote branch ghc-7.4 not found in upstream origin, using HEAD instead
warning: Remote branch ghc-7.4 not found in upstream origin, using HEAD instead
ok.
**** Building stage 1 compiler ... ok.
GHC Version 7.4.1.20120323
**** Building stage 2 compiler ... ok.
**** Building stage 3 compiler ... ok.
**** building testsuite tools ... ok.
**** running tests ... ok (summary below).
**** building compiler binary distribution ... ok.
**** uploading binary distribution ... ok.
**** running nofib (-rtsopts -O2) ... ok.
**** running nofib (-rtsopts -O2 -fllvm) ... ok. (7 failures)
**** running nofib (-rtsopts -O2 -prof -auto-all) ... ok. (2 failures)
**** running nofib (-rtsopts -O2 -prof -auto-all -fllvm)... ok. (1 failures)
**** publishing logs ... ssh: connect to host haskell.org port 22: No route to host
lost connection
failed.
Logs are at http://www.haskell.org/ghc/dist/stable/logs
Dists are at http://www.haskell.org/ghc/dist/stable/dist
Docs are at http://www.haskell.org/ghc/dist/stable/docs

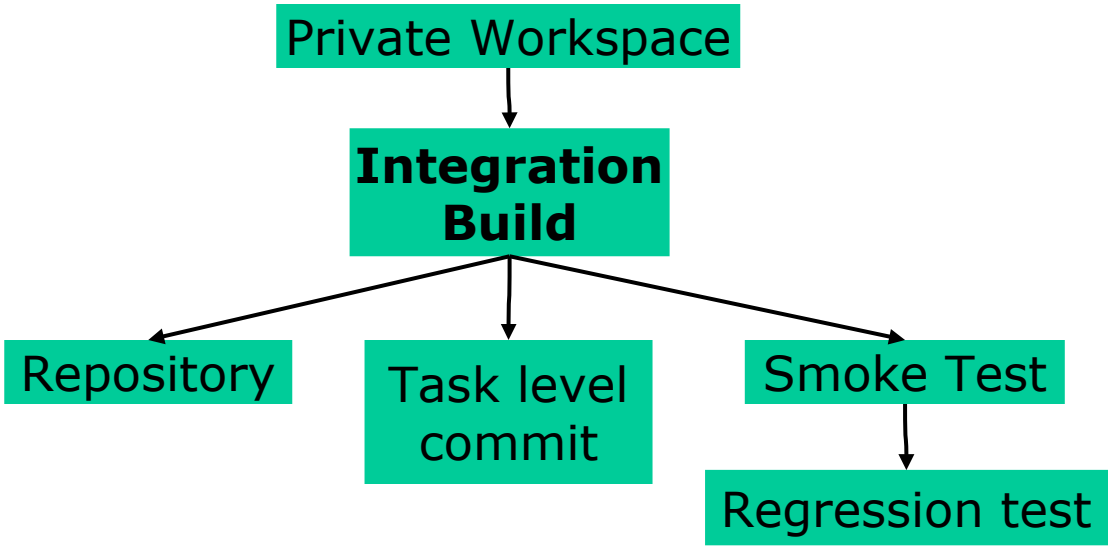
-BR

Integration Build

- Identifique *builds* com uma *tag (Repositório)*
- Repita para todas as plataformas suportadas
- Se um *build* falhar continuamente, adicione mais verificações antes dos commits.
- Se possível, use o Integration Build como base para install kits.

Integration Build (Problemas não resolvidos)

- Alguns padrões complementares
  - Repository: Disponibilizar resultados do build
  - Smoke Test: Testar se o build é útil
  - Regression Test: Testar stable baselines
  - Task Level Commit: Descobrir o que quebrou o build

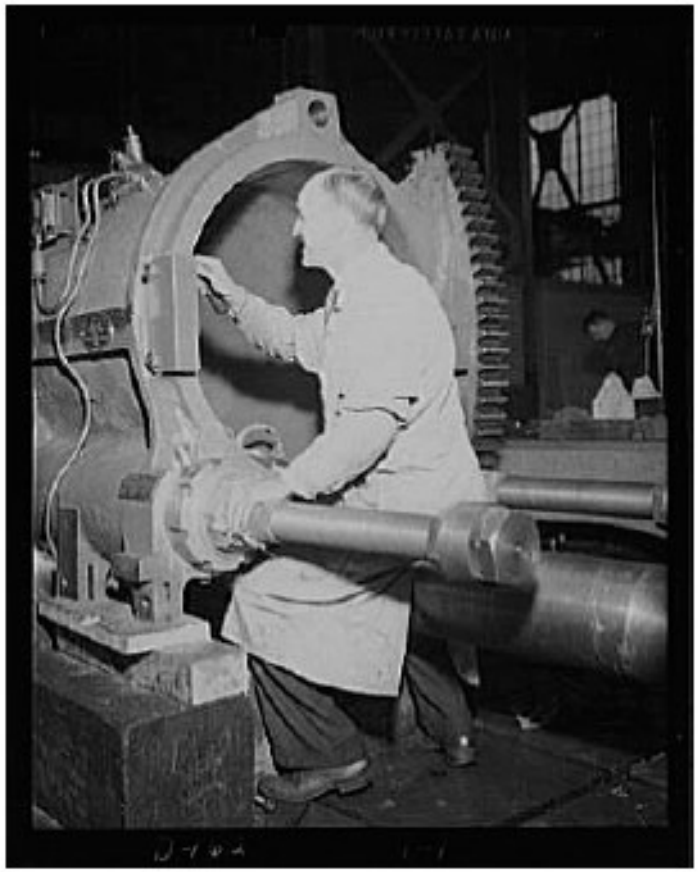


Task Level Commit

"Realize um commit por cada pequena tarefa"

- *Classificação:* Workspace

- Você precisa associar mudanças com um build de integração
- **Quanto você deve trabalhar antes de realizar um commit no Repositório?**



Task Level Commit (Contexto)

- É tentador realizar várias pequenas mudanças em um único commit
  - Políticas rigorosas
  - Tarefas complexas

- Quanto menores as mudanças associadas a um commit,
  - 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 das integrações
  - 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*, garanta que o *workspace* está atualizado com a codeline
- Utilize testes (Unit Test) para assegurar mudanças consistentes

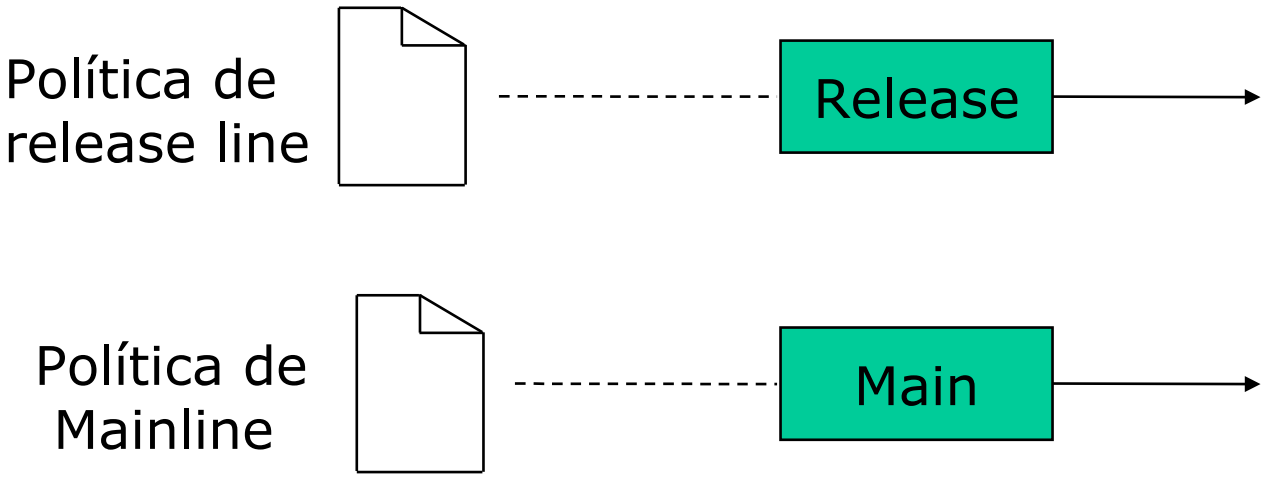
Codeline Policy
"Defina as regras"

- *Classificação:* Codeline

- Active Development Line e Release Line precisam ter regras diferentes.
- **Quando os desenvolvedores sabem quando e como usar cada codeline?**

Codeline Policy (Contexto)

- Diferentes *codelines* têm 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 realizam 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
  - A duração dos trabalhos
  - A carga de trabalho esperada e a freqüência de integrações

Codeline Policy (Boas práticas)

- Textos curtos e objetivos: “direto ao ponto”
  - Com apenas o essencial ao contexto da codeline
  - De 1 a 3 parágrafos. Uma página no máximo
  - Adicione aos comentários de sua *codeline* no seu sistema de controle de versão

Codeline Policy (Boas práticas)

- Textos curtos e objetivos: “direto ao ponto”
  - Com apenas o essencial ao contexto da codeline
  - De 1 a 3 parágrafos. Uma página no máximo
  - Adicione aos comentários de sua *codeline* no seu sistema de controle de versão
- Crie um Branch, sempre que tiver incompatibilidade de políticas
- Divulgação efetiva das políticas no grupo
- Empregue ferramentas automáticas que facilitem o uso de sua política (ex.: ANT)

Smoke Test
"Verifique funcionalidades básicas"

- *Classificação:* Workspace

- Você precisa de verificações em seus builds de forma a manter a Active Development Line.
- **Como verificar que o sistema continua funcionando após a mudança?**

Smoke Test (Contexto)

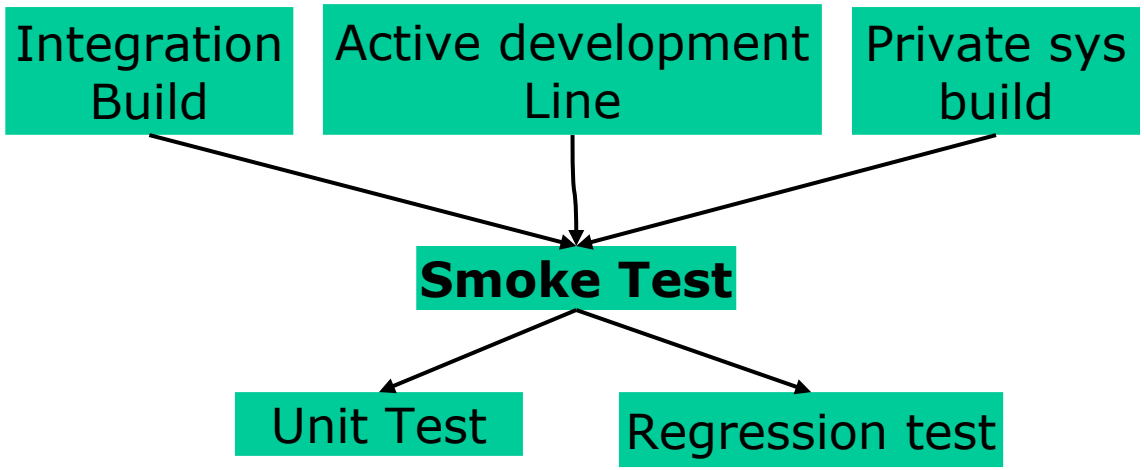
- Testes exaustivos são bons para garantir a qualidade
- Quanto mais longo for teste, mais longos serão os commits
  - Commits com menor freqüência
  - Maior probabilidade de mudanças na Baseline desde a última verificação

Smoke Test (Solução)

- Condicione cada build a um teste de fumaça que verifique que o sistema ainda funcionará em cenários básicos.
  - Testes simples de caixa preta
  - Interações inadvertidas, bugs conhecidos
  - Garante integração básica dos componentes
  - Rápido de executar
  - Provê cobertura geral do sistema

Smoke Test (Problemas não resolvidos)

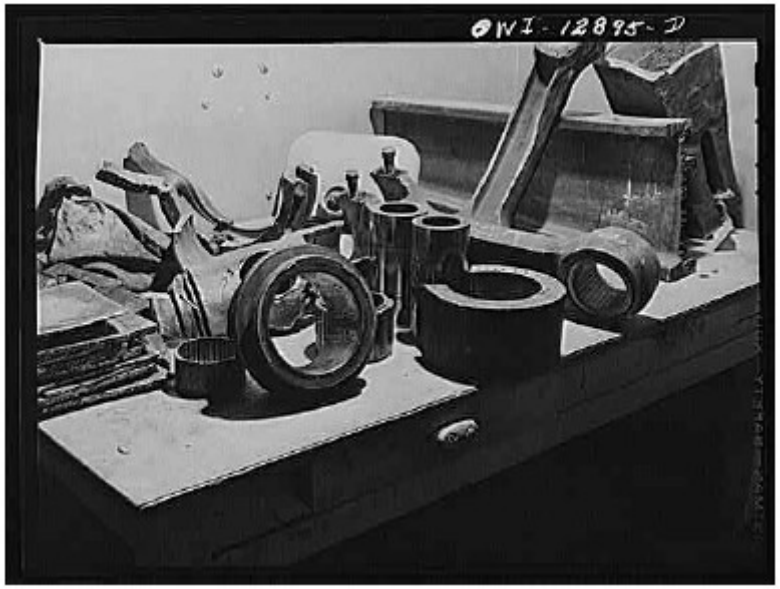
- Smoke Tests não são o suficiente:
  - Regression Test: Lida com problemas que você achava já ter resolvido
  - Unit Test: Nível de abstração menor



Unit Test
"Verifique os contratos"

- *Classificação:* Workspace

- Smoke Tests não são suficientes para verificar que um modulo funciona em um nível menor de detalhes.
- **Como verificar que um módulo continua funcionando após a mudança?**



Unit Test (Contexto)

- Integração identifica problemas, mas não aponta para módulos responsáveis
- Necessidade de isolar questões de integração de questões de mudanças locais das unidades
  - Testar o contrato de cada elemento localmente
  - Estratégia incremental: a parte, depois o todo

Unit Test (Solução)

- **Desenvolva e execute testes unitários**
  - Verificando se um componente obedece seu contrato
- Bons testes unitários são:
  - Simples para rodar
  - Granularidade fina. Cada método significativo da interface de uma classe deve ser testado
  - Automáticos com auto-avaliação. Olha-se para os testes somente quando houver uma falha
  - Isolados. Um teste unitário não interage outros testes

Unit Test

- Testes unitários devem ser executados:
  - Enquanto o software é codificado;
  - Antes de um check-in e após atualizar seu workspace com a versão corrente do software
- Boas práticas:
  - Uso de um framework para escrever testes (Junit, CPPUnit, etc)
  - Redução da quantidade de depuração em prol dos testes unitários. Ganhos de produtividade

Regression Test
"Teste as mudanças"

- *Classificação:* Workspace

- Smoke Tests são bons mas não abrangem todos os casos
- **Como garantir que o programa não se deteriora ao longo do tempo?**

Regression Test (Contexto)

- Abranger todos os casos leva tempo
- É uma boa estratégia adicionar testes à medida que erros são encontrados
- Quando um antigo erro reaparece: você quer estar apto a identificar quando isto acontece
  - mostrar que o software não apresenta problemas dos passado

Regression Test (Solução)

- Desenvolva testes de regressão baseados em situações onde o sistema falhou no passado
  - Prefira testes que envolvam entradas do sistema
- Rodar a cada check-in pode custar caro
  - Unit Tets podem ser mais baratos
- Rode Regression Tests sempre que quiser validar a estabilidade o sistema
  - Antes de um build de release
  - Antes de uma alteração de alto risco

Private Versions
"*Histórico Privado*"

- *Classificação:* Codeline

- Uma Active Development Line ficará inconsistente se commits forem realizados no meio de uma tarefa.
- **Como realizar tarefas complexas e ainda ter os benefícios de um gerenciamento de versões?**

Private Version (Contexto)

- Algumas vezes deseja-se criar um *checkpoint* durante passos intermediários de uma tarefa longa e complexa.
  - Checkpoint é mecanismos oferecido por sistemas de gerenciamento de versões
  - Você não deseja publicar passos intermediários

Private Version (Solução)

- Forneça aos desenvolvedores mecanismos com checkpointing de mudanças
  - Com uma interface simples
  - Na granularidade que eles estejam confortáveis
- Podem ser implementados como
  - Private History
  - Private Repository
  - Private Branch

Task Branch
"*Lidando com tarefas de longa duração*"

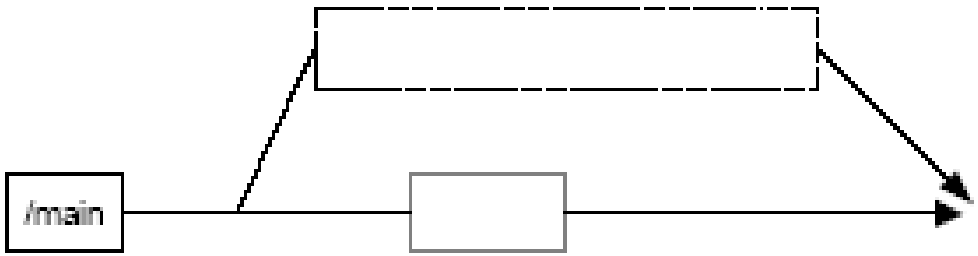
- *Classificação:* Codeline

- Algumas tarefas possuem passos intermediaários que prejudicariam o ritmo de uma Active Development Line
- **Como um time pode realizar múltiplas tarefas de longa duração**
  - **Sem comprometer integridade da codeline?**

Task Branch (Contexto)

- Às vezes somente parte da equipe está trabalhando em uma tarefa
  - A tarefa pode incluir vários passos

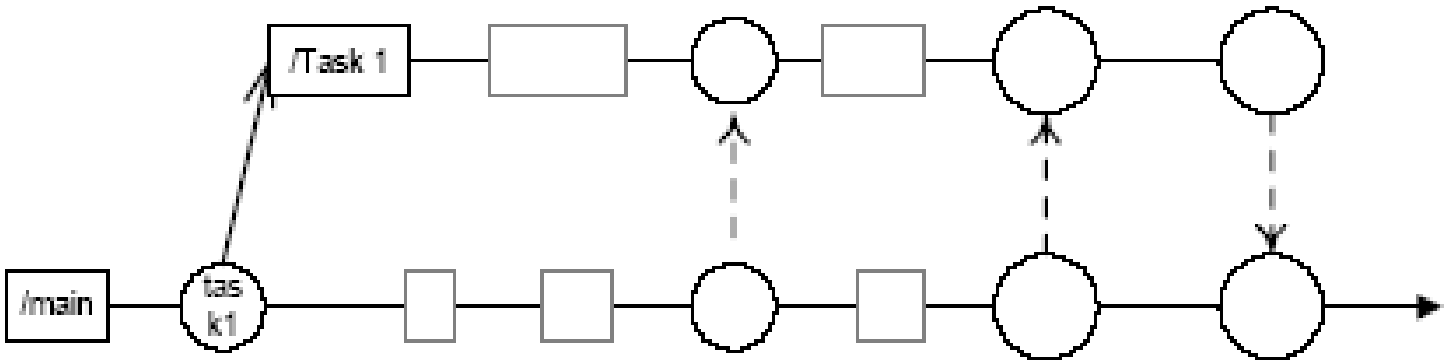
- Exemplo:
  - Grandes *refactorings* não são facilmente realizados em estágios. Ex.: *novo mecanismo de persistência*



Trabalhos para o futuro

Task Branch (Solução)

- Use *Branches* para isolamento da *Mainline*
  - Crie um *branch* para cada atividade que tiver mudanças significativas para a *codeline*.
  - Mecanismo para reduzir riscos
- Para facilitar o *merge final do branch*,
  - **integre regularmente** ao branch as mudanças feitas à Mainline



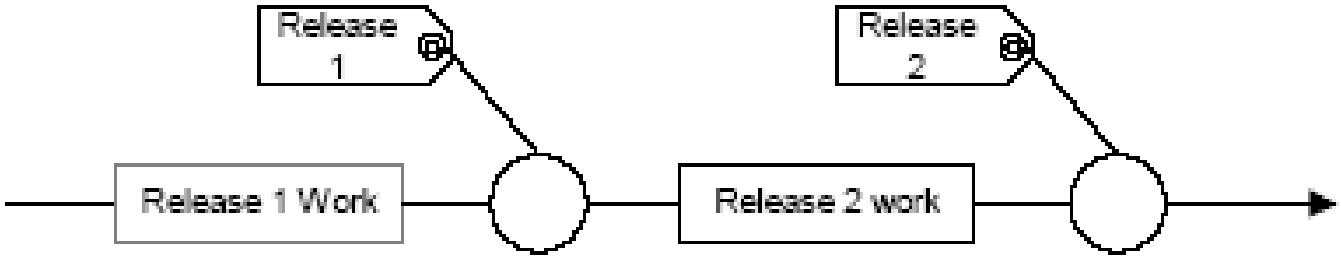
Release Line
"Desenvolvimento Não-Linear"

- *Classificação*: Codeline

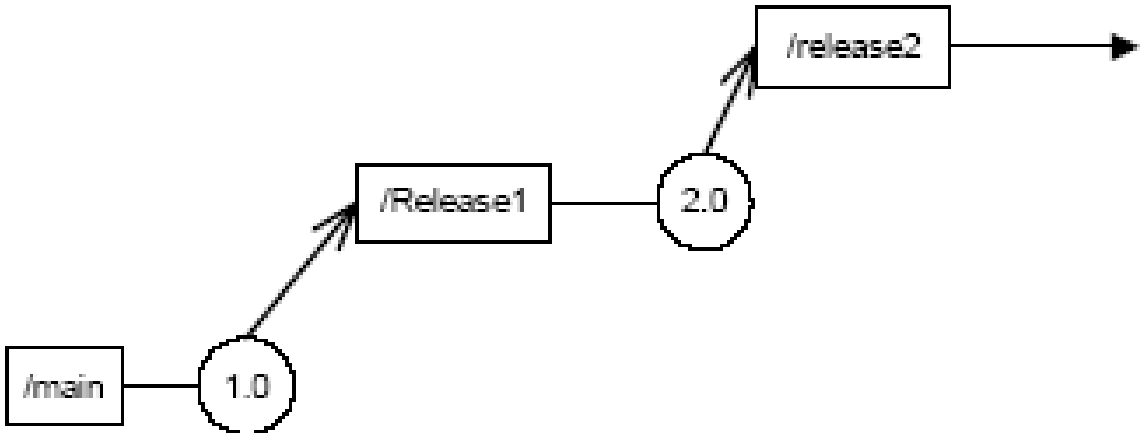
- Você quer manter uma Active Development Line.
- **Como manter uma versão de release sem interferir no trabalho corrente?**

Release Line (Contexto)

- Nem tudo pode evoluir na *mainline*
  - Release precisa de uma política com mais estabilidade

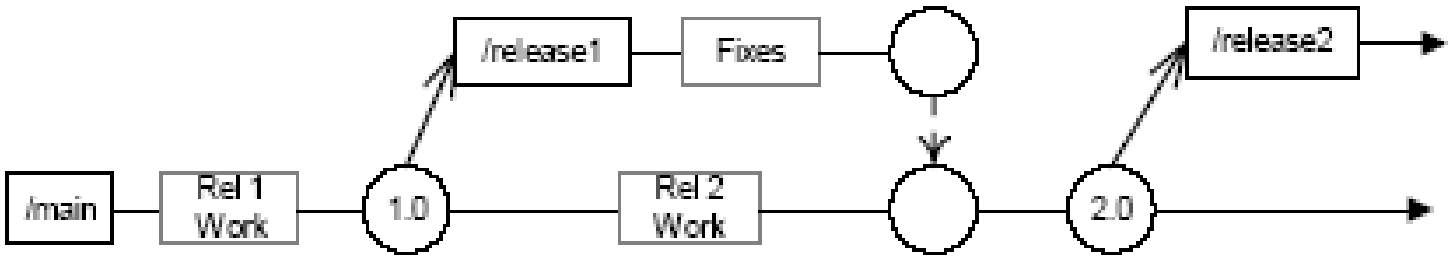


- Precisa-se evitar estruturas complexas



Release Line (Solução)

- Divida manutenção/*release* e desenvolvimento ativo em *codelines* separadas.
- Mantenha cada *release* em uma *release line* independente para possibilitar a correção de *bugs*
- Um *branch* para cada *release line* a partir da **sua** *mainline*.



Release-Prep Codeline
"Evite congelar o código"

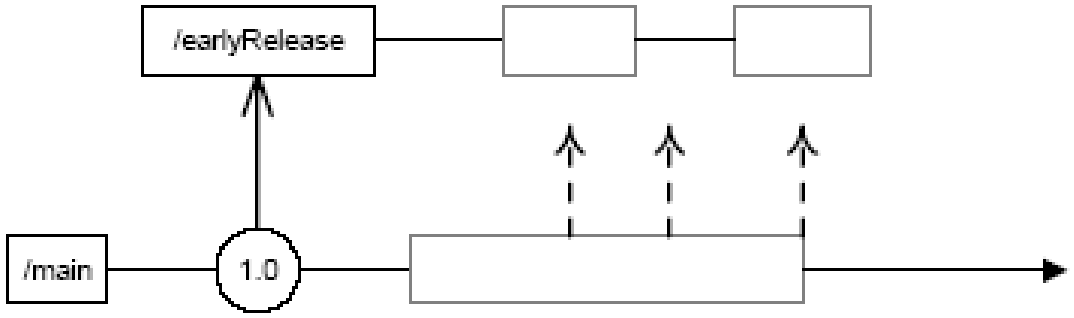
- *Classificação*: Codeline

- Você quer manter uma Active Development Line.
- **Como estabilizar uma codeline para um release iminente ainda permitindo que novas tarefas continuem na codeline ativa?**

Release-Prep Codeline (Contexto)

- A codeline precisa ser estabilizada para que o release seja terminado
  - Congelar novos trabalhos na codeline é penoso

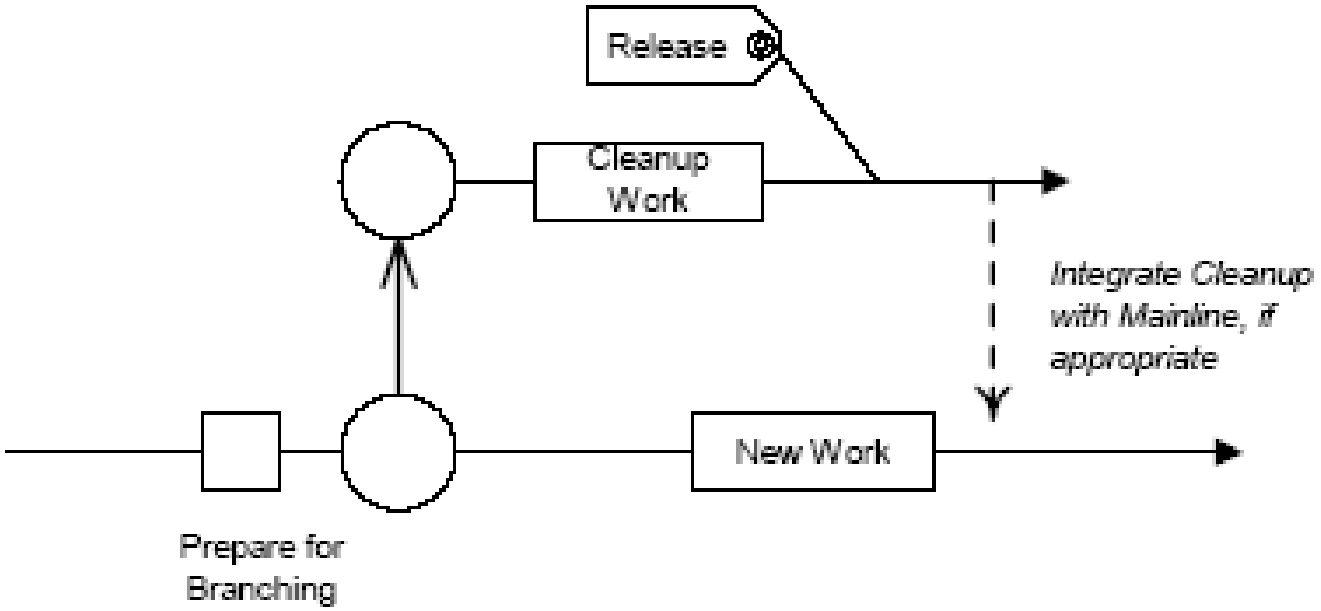
- Uma Release Line pode ser custosa, se apenas uma pequena parte do time está trabalhando em um novo release



Release line antecipada

Release-Prep Codeline (Solução)

- Crie um novo *branch* quando o código estiver se aproximando da qualidade desejada do release.
  - Finalize o *release* neste *branch*, e torne-o o ***Release Line***



A relação entre os padrões