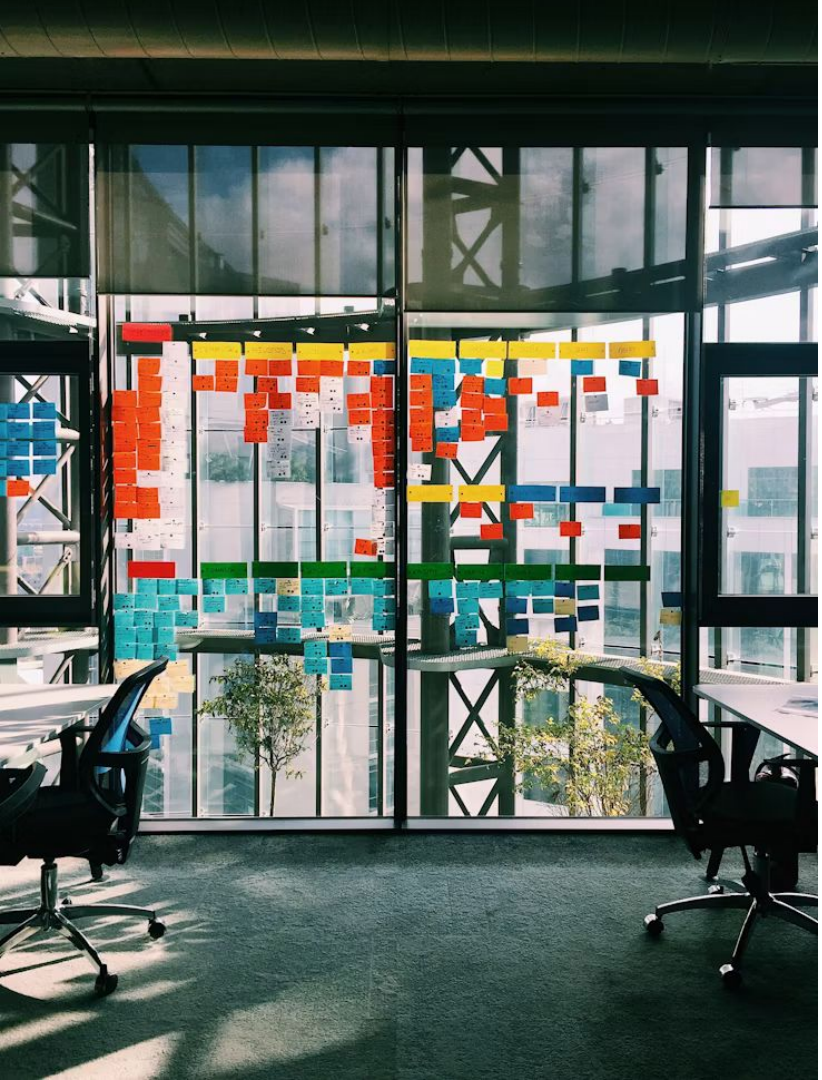


5954014 - Engenharia de Software

Aula 15 - Gestão da Configuração e DevOps

Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Objetivos de Aprendizagem

Compreender a GCS/SCM como disciplina indispensável para controlar a evolução e os artefatos de software.

Identificar os elementos clássicos de SCM: Itens de Configuração (SCIs), Baselines e o Processo Formal de Controle de Mudanças.

Analisar a cultura DevOps, o modelo CALMS e a eliminação do "muro da confusão" entre Desenvolvimento e Operações.

Diferenciar tecnicamente Integração Contínua (CI), Entrega Contínua (Continuous Delivery) e Implantação Contínua (Continuous Deployment).

Mudança Contínua

> ***"Não importa em que ponto do ciclo de vida você esteja, o sistema vai mudar, e o desejo de mudá-lo persistirá por todo o ciclo de vida."***
>
> Primeira Lei da Engenharia de Software (Pressman)

FONTES GERADORAS DE MUDANÇA

01

Mercado & Negócio

Novas regras organizacionais, concorrência e demandas de usuários.

02

Tecnologia & Infraestrutura

Atualizações de plataformas, bibliotecas, hardware e requisitos de segurança.

03

Defeitos e Requisitos

Erros operacionais (bugs) e evolução de funcionalidades existentes.

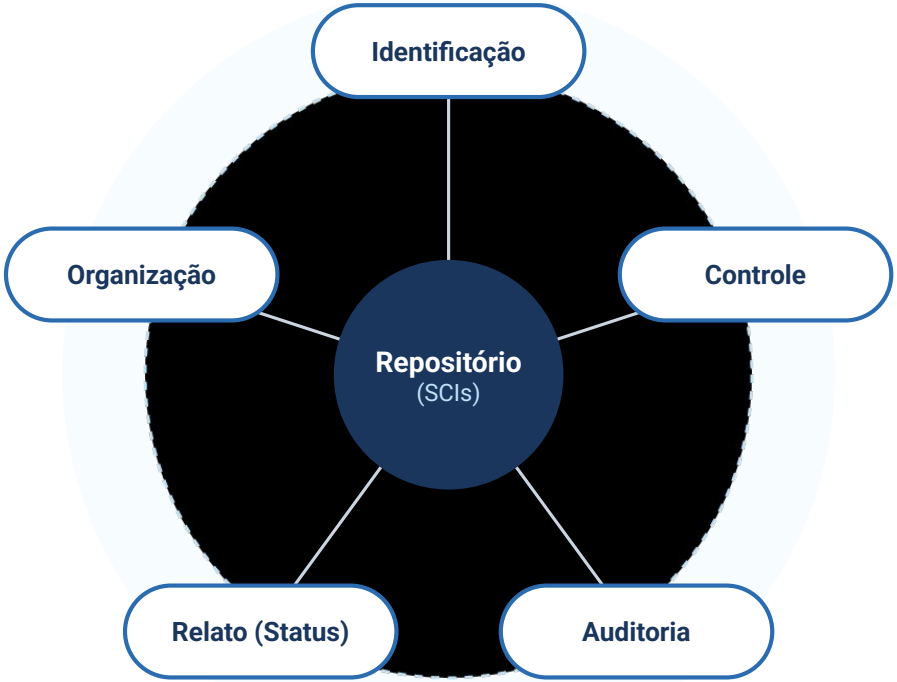
Gestão de Configuração (GCS/SCM)

Conceito (IEEE / Pressman)

Conjunto de disciplinas e atividades de engenharia projetadas para identificar, organizar, controlar, auditar e relatar as alterações feitas nos artefatos de software ao longo do seu ciclo de vida.

PRINCIPAIS OBJETIVOS

- **Rastreabilidade:** Saber exatamente quem alterou o quê, quando e por quê.
- **Integridade:** Manter a qualidade e consistência dos artefatos liberados.
- **Reprodutibilidade:** Reconstruir com precisão qualquer versão anterior do sistema.
- **Minimização de Riscos:** Reduzir custos e conflitos de modificações paralelas.



Itens de Configuração de Software (SCIs)

O que é um Software Configuration Item (SCI)?

Qualquer documento, código, modelo ou dado produzido durante a Engenharia de Software que seja colocado sob controle formal de configuração.

Documentos de Projeto

Artefatos conceituais e estruturais do sistema.

- **Especificação (SRS)**
- **Modelos UML**
- **Dicionário de Dados**

Código & Binários

Elementos executáveis e fontes de software.

- **Código Fonte (.py, .java)**
- **Bibliotecas (.dll, .jar)**
- **Scripts de Build**
- **Executáveis/Binários**

Suíte de Testes

Garantia de qualidade e validação de requisitos.

- **Planos de Teste**
- **Scripts de Automação**
- **Datasets de Teste**
- **Relatórios de Cobertura**

Infra & Operação

Artefatos para implantação e suporte do sistema.

- **Dockerfiles/Helm**
- **Scripts IaC (Terraform)**
- **Arquivos de Configuração**
- **Manuais de Operação**

Repositório de Itens de Configuração

O que é o Repositório de GCS?

Base de dados centralizada (ou distribuída) onde todos os artefatos de software (SCIs), suas versões, metadados e histórico de mudanças são armazenados e gerenciados com segurança.

NÍVEL 1: PRIVADO

1. Biblioteca de Desenvolvimento

Área de trabalho dinâmica e irrestrita do desenvolvedor. Contém código em criação e rascunhos sem controle formal rígido.

ESTADO: DINÂMICO

NÍVEL 2: INTEGRAÇÃO

2. Biblioteca Master (Mestra)

Repositório oficial de Baselines. Armazena SCIs formalmente revisados e aprovados. Alterações exigem autorização (CR).

ESTADO: CONTROLADO

NÍVEL 3: PRODUÇÃO

3. Biblioteca de Arquivo

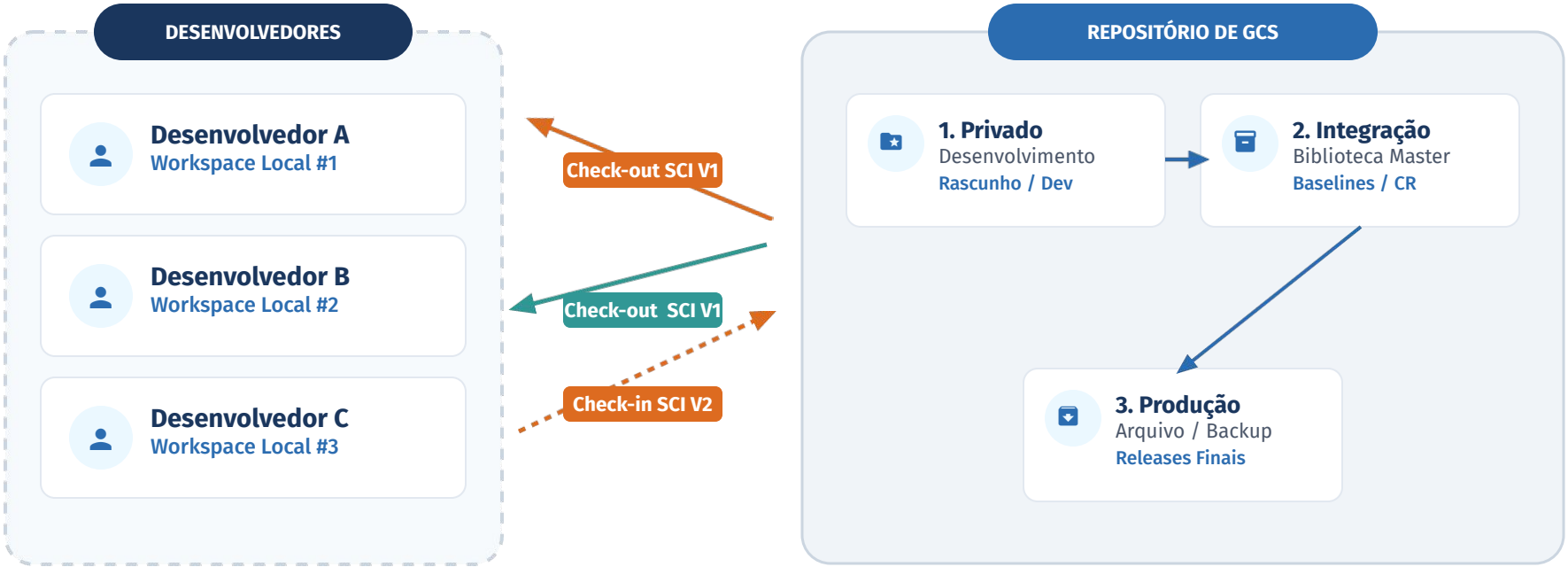
Cópia de segurança permanente (backup/histórico). Preserva todas as versões e releases anteriores para auditoria e recuperação.

ESTADO: PERMANENTE

Fluxo Colaborativo no Repositório de GCS

Colaboração Concorrente & Gestão de SCIs

Desenvolvedores trabalham simultaneamente realizando operações de **Check-out** (extração) e **Check-in / Commit** (integração de SCIs) no repositório centralizado.



Controle de Concorrência: Check-out & Check-in

Check-out (Obter Cópia)

Repositório → Área de Trabalho

- Copia uma versão específica de um SCI do repositório para o espaço de trabalho local do desenvolvedor.
- Permite realizar alterações, testes e experimentações isoladamente sem afetar a linha principal.
- Dependendo do modelo, pode aplicar uma trava (lock) ou sinalizar reserva de edição.

FLUXO DE LEITURA E EDIÇÃO LOCAL

Check-in / Commit (Entregar)

Área de Trabalho → Repositório

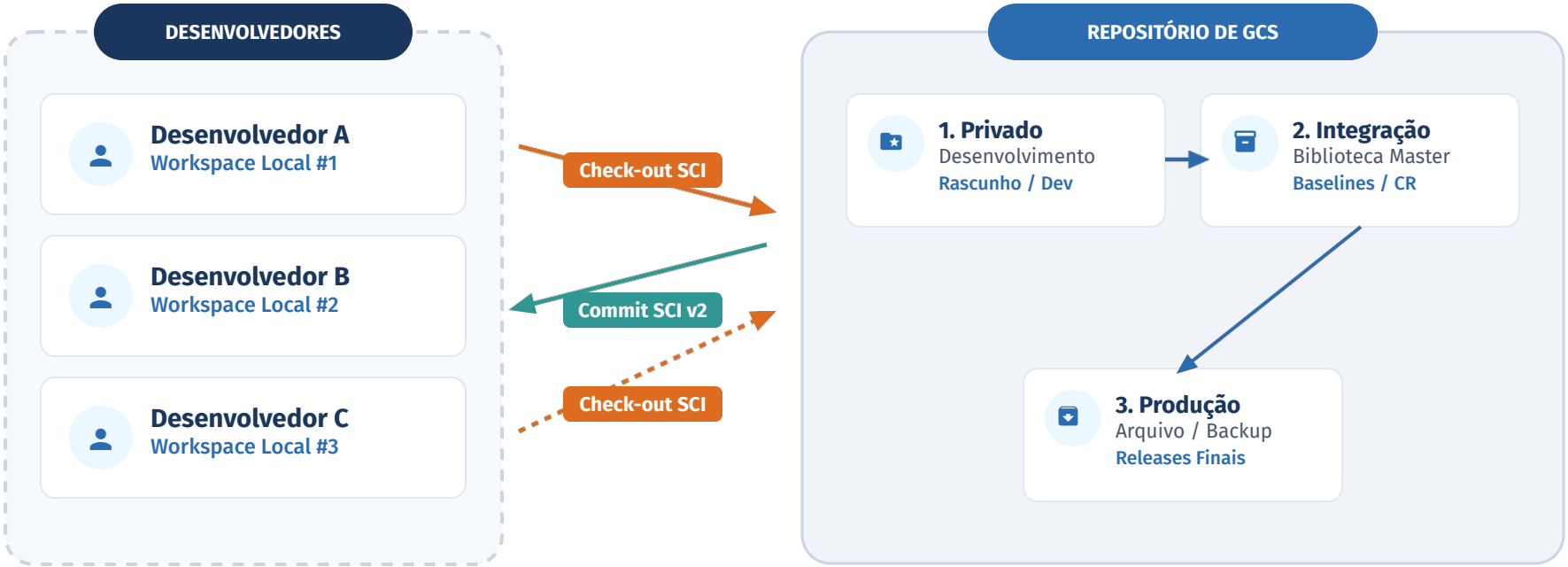
- Devolve o artefato modificado ao repositório central ou remoto, criando uma nova versão oficial.
- Registra metadados essenciais: autor da mudança, data/hora e mensagem explicativa (log).
- Garante a integração das alterações e libera eventuais travas existentes no arquivo.

PERSISTÊNCIA E CRIAÇÃO DE NOVA VERSÃO

Fluxo Colaborativo no Repositório de GCS

Colaboração Concorrente & Gestão de SCIs

Desenvolvedores trabalham simultaneamente realizando operações de **Check-out** (extração) e **Check-in / Commit** (integração de SCIs) no repositório centralizado.



5 Funções Essenciais da GCS (IEEE 828)

01

1. Identificação da Configuração

Definição de SCIs, esquemas de nomes e rótulos de versão.

02

2. Controle de Mudanças

Mecanismos formais para solicitar, avaliar, aprovar e implementar alterações.

03

3. Contabilização de Status

Registro e relatórios do histórico de alterações de cada SCI (quem alterou, quando, por quê).

04

4. Avaliação e Auditoria

Verificações funcionais e físicas para garantir conformidade antes de liberações.

05

5. Gestão de Liberação

Empacotamento, distribuição e construção (build) de versões finais.

Conceito de Baseline

Definição Formal (IEEE Std 610.12-1990)

Uma especificação ou produto de software que foi **formalmente revisado e acordado**, servindo como base para desenvolvimentos posteriores, e que só pode ser alterado através de procedimentos formais de controle de mudanças.

01

Estado Dinâmico

Artefato em desenvolvimento ativo e constantes alterações locais pelo desenvolvedor.

RASCUNHO / EM USO

02

Aprovação Técnica

Submetido a validações rigorosas: revisão por pares, testes automatizados ou aprovação do comitê.

03

Baseline Fixada

Trancado no repositório. Marca um marco (milestone) oficial do projeto.

04

Mudanças Futuras

Qualquer modificação posterior exige estritamente uma Solicitação de Mudança (CR).

Evolução das Baselines no Ciclo de Vida

01

Baseline Funcional

Functional Baseline

- Estabelecida após a aprovação da Especificação de Requisitos do Sistema (SRS).
- Define o que o sistema deve fazer (escopo funcional acordado com o cliente).

REQUISITOS DO SISTEMA

02

Baseline Alocada

Allocated / Development

- Estabelecida após o projeto de arquitetura e design dos componentes.
- Aloca requisitos a módulos de software específicos e interfaces.

ARQUITETURA E DESIGN

03

Baseline de Produto

Product Baseline

- Estabelecida ao final do desenvolvimento e testes formais (Alpha/Beta/Release).
- Contém código-fonte final, executáveis, documentação e manuais prontos.

VERSÃO COMERCIALIZÁVEL

Processo de Controle de Mudança

01

Reconhecimento da Necessidade
Usuário ou desenvolvedor identifica a necessidade de alteração ou falha a ser corrigida.

02

Submissão da CR
Documentação detalhada da alteração solicitada (Change Request) no sistema formal.

03

Avaliação de Impacto
Engenheiros avaliam os riscos técnicos, custos, esforço e prazos da mudança.

04 PONTO DE DECISÃO

Decisão do CCB
O Comitê de Controle de Mudanças (CCB) analisa e decide:

✓ Aprovar
✗ Rejeitar
⏳ Postergar

05

Execução & Checkout
Atribuição da tarefa, checkout dos SCIs da biblioteca e implementação da alteração.

06

Auditoria & (Nova) Baseline
Testes de validação, check-in dos SCIs e incorporação na próxima baseline oficial.

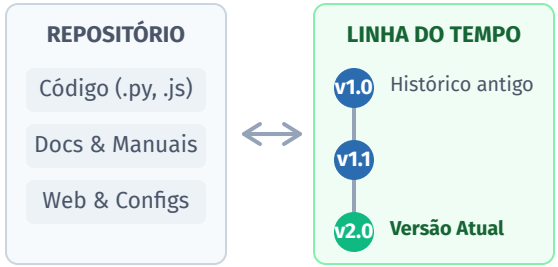
Sistema de Controle de Versões (VCS)

01 Repositório Central & Histórico

Um VCS oferece um repositório para armazenar a **versão mais recente** do código-fonte e arquivos relacionados (documentação, configurações, páginas Web, manuais).

Permite também **recuperar versões mais antigas** de qualquer arquivo historicamente, garantindo rastreabilidade e segurança.

02 Esquema do Repositório (VCS)



03 Esquema de Versionamento: Major . Minor . Revision (SemVer)

v 2 . 4 . 1

MAJOR (Principal)

Alterações grandes e incompatíveis. Mudanças na arquitetura ou quebra de compatibilidade com versões anteriores.

MINOR (Secundário)

Adição de novas funcionalidades mantendo a compatibilidade com as versões anteriores já existentes.

REVISION / PATCH (Revisão)

Correção de bugs e pequenos ajustes finos sem impacto em funcionalidades ou compatibilidade.

Exemplos de Versionamento Semântico

01 1. Major (Principal)

DE 2.0.0 PARA 3.0.0

Mudança Incompatível

O Que é: Mudanças significativas e incompatíveis com versões anteriores.

Quando Atualizar: Alterações drásticas, como remoção de recursos ou quebra de API.

Impacto: Requer adaptação do sistema antes de atualizar.

02 2. Minor (Funcionalidade)

DE 2.3.0 PARA 2.4.0

Nova Feature Compatível

O Que é: Lançamento de novas funcionalidades mantendo compatibilidade.

Quando Atualizar: Novas features são adicionadas sem remover recursos antigos.

Impacto: Atualizações seguras para aplicações existentes.

03 3. Patch (Correção)

DE 2.3.4 PARA 2.3.5

Correção de Bugs

O Que é: Pequenas correções de bugs ou ajustes de segurança.

Quando Atualizar: Correções internas sem novas funcionalidades.

Impacto: Atualizações simples e seguras, focadas na manutenção.

Auditorias de Configuração (FCA vs. PCA)

Auditoria Funcional (FCA)
Functional Configuration Audit

FOCO DA AUDITORIA

Validação do Comportamento

PERGUNTA-CHAVE

"O software realmente executa todas as funções especificadas nos requisitos?"

MÉTODOS DE VERIFICAÇÃO

- Execução de suítes de testes automatizados e manuais.
- Condução de testes de aceitação com usuários/clientes.
- Análise detalhada dos relatórios de validação de cobertura.

Auditoria Física (PCA)
Physical Configuration Audit

FOCO DA AUDITORIA

Verificação dos Artefatos

PERGUNTA-CHAVE

"Todos os SCIs (código, documentação, scripts) estão presentes, corretos e formatados?"

MÉTODOS DE VERIFICAÇÃO

- Inspeção física de arquivos, documentação e manuais.
- Auditoria da estrutura do repositório e versionamento.
- Validação de compilação limpa (clean build) do zero.

Gerenciamento de Configuração de Software oferece um ambiente de trabalho estável.

Processos e uma "**memória**" do estado dos produtos de desenvolvimento de software.

Git

O Que é

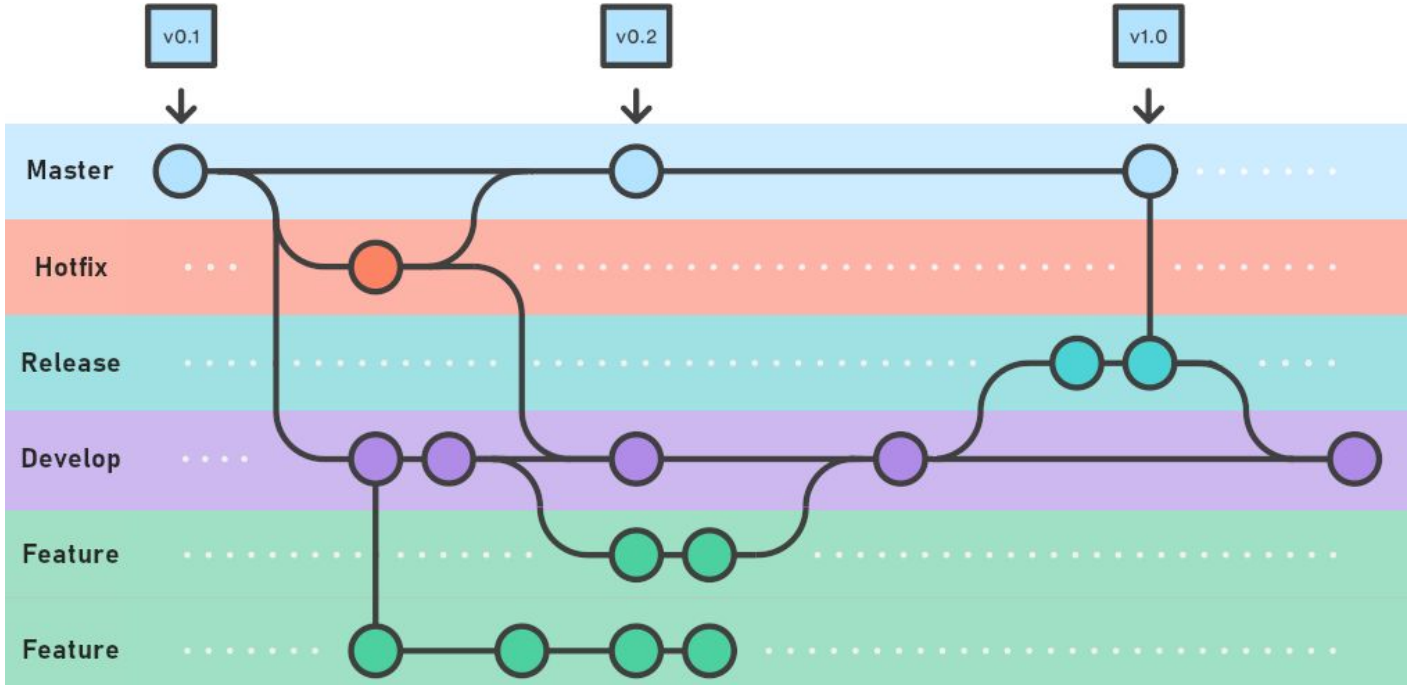
Sistema VCS distribuído que registra commits, ramos (branches) e tags de baselines.

Aplicação GCS

Armazena o histórico completo de SCIs (código, scripts e docs).

Benefício

Permite auditoria, restauração de versões passadas e trabalho em paralelo.



Git Flow

`main` **Estável:** Versões de produção (tags v1.0).

`develop` **Integração:** Base para novas funcionalidades.

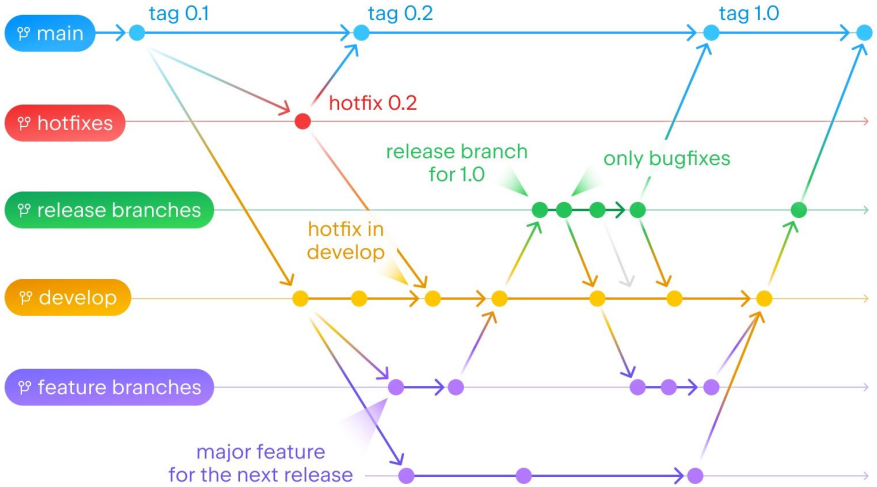
`feature/*` **Recursos:** Criada de `develop`. Reintegra a ela após conclusão.

`release/*` **Preparação:** Criada de `develop`. Reintegra em `main` e `develop`.

`hotfix/*` **Urgente:** Criada de `main` para correções em produção.

Exemplo: Reserva de Salas

1. `v1.0` está em produção na branch `main`.
2. Cria-se `feature/cancelamento` a partir de `develop`.
3. Funcionalidade concluída é integrada à `develop`.
4. Cria-se `release/1.1` para preparar a nova versão.
5. Após validação, integra em `main` (tag `v1.1`) e `develop`.
6. Erro crítico em produção? Cria-se `hotfix/*` de `main`.



- The `main` branch is for production code only.
- The `develop` branch is for development code.
- `feature` branches are created from the `develop` branch.
- `hotfix` branches are created from the `main` branch.
- `release` branches are created from the `develop` branch.

Fonte: https://blog.jetbrains.com/space/2023/04/18/space-git-flow/

Trunk-based Development

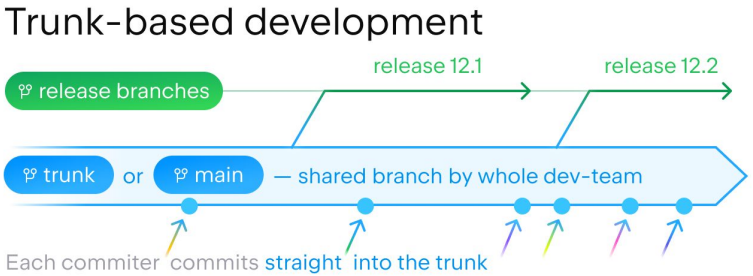
Branch Principal (`trunk/main`)
Concentra o código da equipe e deve estar sempre pronta para produção.

Integração Frequente
Devs integram pequenas alterações continuamente (várias vezes ao dia).

Branches Curtas & Releases
Branches de curta duração reduzem divergências; releases saem da **`main`**.

Testes Automatizados & CI
Validação por pipeline de CI antes e após cada integração para garantir estabilidade.

Colaboração Contínua
Compartilhamento frequente de código minimiza conflitos complexos de merge.



- All code is committed to a single branch, usually called `trunk`.
- The `trunk` branch is always production-ready.
- Development work is done directly on the `trunk` branch.
- There aren't any long-lived `feature` branches.
- Before pushing changes, developers perform a full pre-integration build (compile, unit tests, etc.) on their local machines.
- After the changes are already in `trunk`, there may be a post-integration code review.

Fonte: https://blog.jetbrains.com/space/2023/04/18/space-git-flow/

Exemplo de GCS em Projeto Python

Aplicação da Gestão de Configuração de Software (SCIs + Versionamento)
Controle de artefatos de um script de automação (app.py) e suas dependências.

1. ITENS DE CONFIGURAÇÃO (SCIs)

Código & Dependências
- main.py (Script Principal)
- requirements.txt (Libs)

Ambiente & Infra
- .env.example (Variaveis)
- Dockerfile (Container)

Testes & Qualidade
- test_main.py (Pytest)
- pytest.ini (Configs)

Não Rastreados (.gitignore)
- venv/, __pycache__/, .env

2. FLUXO DE CONTROLE DE MUDANÇAS (GIT / BASELINE)

```
# 1. Definir Baseline Inicial (v1.0.0)
git init && git add main.py requirements.txt
git commit -m "feat: baseline inicial do projeto"
git tag -a v1.0.0 -m "Release v1.0.0"
```

```
# 2. Mudança Controlada (Solicitação/Bugfix)
git checkout -b fix/issue-42-conexao-db
# ... alterações nos SCIs requeridos ...
git commit -m "fix: corrige timeout no banco de dados"
git push origin fix/issue-42-conexao-db
```

```
# 3. Auditoria & Nova Baseline (PR / Merge)
# Revisa-se a rastreabilidade e executa testes (CI)
git checkout main && git merge fix/issue-42-conexao-db
git tag -a v1.0.1 -m "Release v1.0.1 (Patch)"
```

A Origem do DevOps

DEV

Desenvolvimento

Foco em Agilidade e Mudança

- Pressionado para entregar novas funcionalidades e inovações rapidamente.
- Foco em promover mudanças frequentes e velocidade de liberação.

O CONFLITO TRADICIONAL

O "Muro da Confusão"

"Dev joga o código por cima do muro..."

...e Ops sofre isoladamente para implantar e manter o software rodando em produção sem quedas.

OPS

Operações

Foco em Estabilidade e Risco

- Pressionado para manter o ambiente estável, disponível e sem falhas.
- Evita alterações de risco para garantir o SLA e a segurança operacional.

∞

A SOLUÇÃO INTEGRADORA

A Abordagem DevOps

- **Extensão Ágil:** Leva as práticas ágeis até a "última milha" do software (implantação em produção).
- **União Cultural:** Integração, responsabilidade compartilhada e automação contínua do fluxo de valor.

Pilares do DevOps: O Modelo CALMS

C

Culture

Cultura

Colaboração, alinhamento de metas, empatia e eliminação de silos operacionais.

A

Automation

Automação

Automação de builds, testes, provisionamento de infraestrutura e deployments.

L

Lean

Enxuto

Redução de desperdícios, lotes pequenos de trabalho e melhoria do fluxo de valor.

M

Measurement

Medição

Monitoramento contínuo de métricas de negócio, desempenho e frequência de deploys.

S

Sharing

Compartilhamento

Transparência de conhecimento, aprendizado com falhas e post-mortems sem culpa.

Princípios de Entrega Confiável (Humble & Farley)

01 **Processo Repetível e Confiável**
A entrega de software em produção deve ser um evento rotineiro e previsível.

02 **Automatize Tudo**
Do *build* e testes à configuração de rede e provisionamento de servidores.

03 **Tudo Sob Controle de Versão**
Código, scripts de infraestrutura, bancos de dados, testes e documentação.

04 **Se Dói, Faça Com Mais Frequência**
Antecipe dores. Integrar ou implantar com frequência reduz o tamanho do lote e o risco.

05 **Responsabilidade Compartilhada**
"Se houver um problema em produção, a equipe inteira é responsável."

Anatomia de um Pipeline de Integração Contínua (CI)

01

Commit & Push

Desenvolvedor envia o código para o repositório central (git push).

02

Disparo Automatizado

O repositório notifica o servidor de CI (GitLab CI/Jenkins) via *Webhook*.

03

Compilação (Build)

O servidor de CI baixa o código e realiza o *build* limpo do projeto.

04

Testes Automatizados

Execução de testes unitários, de integração e análise estática de código (ex.: SonarQube).

05

Feedback

Notificação imediata em caso de falha ou sucesso.

REGRA DE OURO DO CI

"Corrigir o build quebrado é a prioridade #1 da equipe."

Se o pipeline quebrar, o trabalho para! Ninguém faz novos commits até que a integração seja restabelecida.

Continuous Delivery vs. Continuous Deployment

A base comum é a Integração Contínua (CI): Código integrado, construído e testado automaticamente a cada alteração.

Continuous Delivery (Entrega Contínua)

Software sempre em estado de liberação (releaseable).

A promoção técnica é 100% automatizada, mas exige uma **decisão de negócio / clique manual** de aprovação.

Cenários Típicos:

- Softwares médicos
- Sistemas bancários
- Aplicativos móveis
- Drivers de impressora

Continuous Deployment (Implantação Contínua)

Automação Ponta a Ponta:

Cada commit aprovado vai **diretamente para produção** sem intervenção humana.

Cenários Típicos:

- SaaS web
- Plataformas de e-commerce
- Serviços em nuvem

Containerização: Eliminando o "Funciona na Minha Máquina"

Problema Histórico: Diferenças de versões de bibliotecas, variáveis de ambiente e SO entre a máquina do desenvolvedor e o servidor de produção.

Máquinas Virtuais (VMs Tradicionais)

SO Convidado Completo:
Cada VM requer seu próprio SO, consumindo gigabytes de RAM/disco.

Hypervisor:
Camada extra de abstração que gera sobrecarga de CPU e I/O.

Inicialização Lenta:
Minutos para boot e provisionamento.

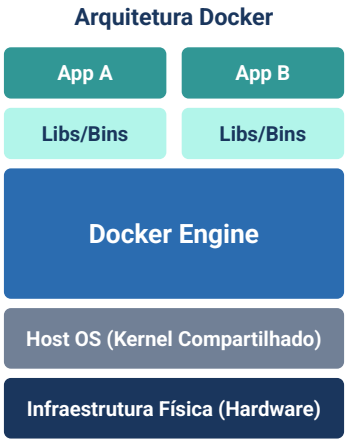


Contêineres Docker (Arquitetura Leve)

Empacotamento Completo:
Código, runtime, dependências e configs em uma imagem imutável.

Kernel Compartilhado:
Isolamento em nível de SO sem carregar um SO convidado.

Alto Desempenho:
Leveza e inicialização em milissegundos.



Infraestrutura Declarativa

Conexão com a GCS: Ambos os arquivos são Artefatos Declarativos (SCIs) versionados e armazenados diretamente no repositório Git do projeto.

`Dockerfile`: Receita da Imagem

```
# Imagem base oficial
FROM python:3.11-slim
# Diretório de trabalho
WORKDIR /app
# Copia e instala dependências
COPY requirements.txt .
RUN pip install -r requirements.txt
# Copia o código da aplicação
COPY . .
# Comando de inicialização
CMD ["python", "app.py"]
```

`docker-compose.yml`: Multi-Serviços

Orquestração Local Multi-Serviços:
Declaração conjunta de ambiente completo de desenvolvimento.

Aplicação Principal (App Python)
Construída a partir do Dockerfile local com reload dinâmico.

Banco de Dados (PostgreSQL)
Persistência de dados com volumes acoplados e credenciais.

Camada de Cache (Redis)
Armazenamento em memória para alta performance de sessão.

Infraestrutura como Código na Prática: Documentos de texto que especificam imagem, dependências e serviços **vão para o repositório** junto com o código-fonte do sistema.

Exercício

CENÁRIO: SOFTWARE CRÍTICO PARA SAÚDE

Uma startup de software médico desenvolve um sistema de apoio a diagnósticos em UTIs. O sistema possui um módulo web e uma API de integração com monitores cardíacos.

1 Estratégia de Deploy

Para o módulo de diagnóstico da UTI, a equipe deve adotar **Continuous Delivery** ou **Continuous Deployment**?

Justifique com base em riscos e conformidade.

2 Branching & Testes

A equipe está sofrendo com merges que levam **3 dias**.

Qual mudança de prática de **branching** e **automação de testes** vocês recomendariam para esse time?

Conclusão

Síntese da Aula

GCS: Garante rastreabilidade, baselines e integridade de todos os artefatos.

DevOps: Canaliza essa disciplina através da automação (**CI/CD**, **IaC** e **Docker**) para entregas rápidas e confiáveis.

Tendências Emergentes

GitOps: O Git como fonte única da verdade para gerenciar infraestrutura e clusters Kubernetes.

DevSecOps: Integração de segurança desde os primeiros estágios da esteira (Shift-Left).

AIOps: Uso de IA/ML para monitoramento e detecção preditiva de anomalias operacionais.

Autoavaliação

Pergunta de Autoavaliação:

"Qual a diferença conceitual entre uma **Promoção** *e uma* **Liberação** *na GCS tradicional, e como elas se traduzem em um pipeline de CI/CD?"*

Resumo: A GCS fornece a disciplina e o rigor de controle, enquanto o DevOps fornece a agilidade e a automação.

Referências

Bibliografia Básica da Disciplina

Roger S. Pressman & Bruce R. Maxim

Engenharia de Software: Uma Abordagem Profissional

Capítulos 19 e 20 (completos)

Ian Sommerville

Engenharia de Software

Capítulo 8 (completo)

Referências Complementares de Apoio

Marco Tulio Valente
Engenharia de Software Moderna
Capítulo 10 (completo)
https://engsoftmoderna.info/cap10.html

Primeiros passos com o Git
Tutorial do GitHub Docs
https://docs.github.com/pt/get-started/learning-to-code/getting-started-with-git

Dúvidas e Discussão